A MULTI-PERSPECTIVE METHODOLOGY FOR AGENT-ORIENTED BUSINESS MODELLING AND SIMULATION

KULDAR TAVETER

Faculty of Information Technology,

Department of Informatics,

TALLINN UNIVERSITY OF TECHNOLOGY

Dissertation is accepted for the commencement of the degree of Doctor of Philosophy in Engineering on .....……., 2004.

Supervisor: Prof Emeritus Boris Tamm (up to 5.2.2002)

Co-supervisor: Prof Dr Gerd Wagner, Chair/Professor in Internet Technologies, Cottbus University, Germany; Assistant Professor of Information Systems, Eindhoven University of Technology, the Netherlands

Opponents: Prof Dr Brian Henderson-Sellers, Professor of Information Systems, Director of the Centre for Object Technology Applications and Research, University of Technology, Sydney, Australia

Dr Frank Dignum, Associate Professor, Institute of Information and Computing Sciences, Utrecht University, the Netherlands

Commencement: 12 June 2004

Declaration

Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted before for any degree or examination.

Signature of candidate:

Date:

ISSN

ISBN

To my parents

To Siiri

To Eliise and Sanne

Abstract

With the development of Internet and communication facilities, the importance of distributed information systems both within a given enterprise and between enterprises is increasing. As business modelling always constitutes the first stage in the lifecycle of an information system, developing such information systems requires a new modelling paradigm that would put more emphasis on the modelling of communication and interaction. Agent-Orientation is emerging as such a new paradigm in business modelling and information systems engineering. Agent-Orientation emphasizes the fundamental role of actors/agents and their mental state, and of communication and interaction, for analyzing and designing organizations and organizational information systems. An agent is thus not just a technological building block (software agent), like it is sometimes understood, but also an important modelling abstraction that can be used at different logical levels in the creation and development of an information system.

In this thesis, we first provide new definitions of business rules and business processes and a classification of business rules that comply with the paradigm of Agent-Orientation. We then make the scope of Agent-Orientation more precise by defining six views of agent-oriented modelling as an improvement on the existing business modelling frameworks. These views are the organizational, informational, interactional, functional, motivational, and behavioural view. Thereafter we evaluate and compare seven "traditional" business modelling techniques and one agent-oriented software engineering technique with respect to the six views and argue for the need for a distinctive technique of agent-oriented business modelling.

The main contribution of this thesis is a technique for agent-oriented business modelling covering all the views mentioned above and producing executable models. This modelling technique, called the Business Agents' Approach, consists of the steps of analysis and design. The analysis is performed by means of goal-based use cases, while the design is based on a combination of the Agent-Object-Relationship Modelling Language (AORML) and the Object Constraint Language (OCL), now forming a part of the Unified Modelling Language (UML). The business modelling technique proposed by us also draws guidelines for transforming goal-based use cases of the step of analysis to multi-perspective models of the design step.

For the step of design, we extend the graphical notation of AORML by activity diagrams and describe how they function. After that, we provide an operational semantics for activity diagrams based on the semantic framework of Knowledge-Perception-Memory-Commitment (KPMC) agents. We also define semi-formally the activity modelling language that serves as a foundation for activity diagrams and outline the way how activity diagrams can be mimicked on the JADE agent platform. After that, we show how activity diagrams enable to combine models of all six views of agent-oriented modelling. Additionally, for creating models of the informational view, we propose some modifications to OCL. In order to represent models of the interactional view, we elaborate on the interaction frame diagrams of AORML.

Finally, we apply the business modelling methodology created by us to the case studies of the ceramic factory and business-to-business electronic commerce in the field of advertising. The goals of the case studies are simulating the business processes of these domains and preparing for their automation.

The Business Agents' Approach thus allows increased capturing of the dynamic and deontic semantics of business modelling in comparison with e.g. object-oriented modelling approaches, such as UML. Taking into account that the main motivation for object-oriented modelling stems from software engineering and not from business modelling, or cognitive modelling, this should not be surprising.

Keywords: agent, actor, business rule, business process, ontology, business modelling, conceptual modelling, enterprise modelling

Kokkuvõte

Interneti ja kommunikatsioonivõimaluste arenguga kasvab nii ettevõttesiseste kui ka ettevõtetevaheliste hajutatud infosüsteemide tähtsus. Kuna ärimodelleerimine on alati infosüsteemi elutsükli esimeseks sammuks, nõuab selliste infosüsteemide loomine uut modelleerimisparadigmat, mis paneks rohkem rõhku kommunikatsiooni ja interaktsiooni modelleerimisele. Niisuguseks uueks paradigmaks ärimodelleerimisel ja infosüsteemide loomisel on agentorienteeritus. Agentorienteeritus rõhutab agentide/tegijate ja nende "vaimsete" olekute ning agentidevahelise kommunikatsiooni ja interaktsiooni fundamentaalset tähtsust organisatsioonide ja nende infosüsteemide analüüsimisel ja kavandamisel. Agent ei ole seega mitte ainult tehnoloogiline ehitusblokk (tarkvaraagent), nagu vahel on aru saadud, vaid ka oluline modelleerimisabstraktsioon, mida saab kasutada infosüsteemi loomise ja arendamise erinevatel loogilistel tasemetel.

Käesolevas väitekirjas anname kõigepealt ärireeglite ja -protsesside uued definitsioonid ning ärireeglite uue klassifikatsiooni, mis kõik on vastavuses agentorienteerituse paradigmaga. Pärast seda täpsustame agentorienteerituse skoopi defineerides olemasolevate ärimodelleerimise raamistike edasiarendusena agentorienteeritud modelleerimise kuus vaadet. Need vaated on organisatsiooniline, informatsiooniline, interaktsiooniline, funktsionaalne, motivatsiooniline ja käitumuslik vaade. Seejärel hindame ja võrdleme seitset "traditsioonilist" ärimodelleerimise menetlusviisi ja ühte agendipõhist tarkvaratehnikat nimetatud kuue vaate suhtes ning põhjendame erilise menetlusviisi vajadust agentorienteeritud modelleerimise jaoks.

Käesoleva väitekirja peamine panus ongi agentorienteeritud modelleerimise menetlusviis, mis hõlmab kõiki ülalnimetatud vaateid ja võimaldab täidetavate mudelite loomist. See menetlusviis, mille nimetuseks on "Äriagentide lähenemisviis", koosneb analüüsi ja kavandamise (disaini) sammudest. Analüüsiks kasutatakse modelleerimist eesmärkidega kasutusjuhtude abil ning kavandamine põhineb graafilise modelleerimiskeele AORML-i ja OCL-i, mis nüüdseks moodustab UML-i osa, kombinatsioonil. Meie poolt väljatöötatud menetlusviis sisaldab ka juhiseid analüüsi tulemusena saadud eesmärkidega kasutusjuhtude teisendamiseks kavandamissammu mitmevaatelisteks mudeliteks.

Kavandamissammu jaoks täiendame AORML-i graafilist tähistusviisi tegevusdiagrammidega ja kirjeldame nende funktsioneerimist. Seejärel defineerime tegevusdiagrammide jaoks operaatorsemantika, mis põhineb KPMC-agentide teoorial. Samuti defineerime poolformaalselt agendi funktsioonide ja käitumise modelleerimiskeele, mis on tegevusdiagrammide aluseks, ning kirjeldame viisi, kuidas tegevusdiagramme saab simuleerida JADE agendiplatvormil. Pärast seda näitame kuidas tegevusdiagrammid võimaldavad agentorienteeritud modelleerimise kõigi kuue vaate mudelite kombineerimist. Sellele lisaks pakume informatsioonilise vaate mudelite loomiseks välja rea OCL-i modifikatsioone ning arendame interaktsioonilise vaate mudelite loomiseks edasi AORML-i interaktsioonidiagramme.

Lõpuks rakendame meie poolt loodud modelleerimismetodoloogiat keraamikatehase juhtumianalüüsile ja ettevõtetevahelise elektronkaubanduse juhtumianalüüsile reklaaminduses. Juhtumianalüüside eesmärkideks on nimetatud valdkondade äriprotsesside simuleerimine ja automatiseerimiseks ettevalmistamine.

"Äriagentide lähenemisviis" võimaldab seega ärimodelleerimise dünaamilise ja deontilise semantika täpsemat esitamist kui näiteks objektorienteeritud modelleerimisviisid nagu UML. Kui võtta arvesse, et objektorienteeritud modelleerimise peamiseks lähtekohaks on tarkvaratehnika, mitte ärimodelleerimine või kontseptuaalne modelleerimine, ei ole see üllatav.

Võtmesõnad: agent, tegija, ärireegel, äriprotsess, ontoloogia, ärimodelleerimine, kontseptuaalne modelleerimine, ettevõtte modelleerimine

Acknowledgements

First of all, I would like to pay reverence to and Thank God for my late supervisor Prof Emeritus Boris Tamm for offering me invaluable advice in the course of my doctoral studies. In addition to his scientific encouragement and support, in the spring and summer of 2001 he even provided me with a corner in his office, so that I could work on my thesis. Unfortunately, it is too late to thank him personally.

I would also like to express the warmest and kindest thanks to my co-supervisor Prof Gerd Wagner from Cottbus University and the Eindhoven University of Technology. His scientific guidance and pressure to finalize the thesis is beyond thanks. It is the results of his previous research work that this thesis builds on.

Add Reet Hääl and her colleagues from Tallinn Ceramic Factory to the list of people that were irreplaceable in helping me get my thesis finished. Thank you all, very much, for enabling me to use the factory as a basis for one of the case studies and for providing me with the source materials for the case study. For the same reason, I extend my thanks to the colleagues Asta Bäck and Hannele Antikainen from the Technical Research Centre of Finland.

I am grateful to the Estonian Science Foundation which has supported the research work that has led to this thesis by its grant number 4721. I would also like to thank the Estonian National Culture Foundation for partial funding of my attendance at the Conference on Conceptual Modelling in Japan. I bow many times towards these two Foundations from Estonia.

If it wasn't for the vacation of three months and numerous unpaid vacations to work on my thesis, complements of the Technical Research Centre of Finland, writing of this thesis would still have been under way. Thank you Technical Research Centre of Finland, for helping me finish the thesis this way.

I would also like to thank Prof Seppo Linnainmaa and Aarno Lehtola for enabling me to attend conferences and other research events related to my thesis. These conferences and events were crucial to my thesis. Also, many thanks to the leading group and team of the Plug-and-Trade B2B research project where the results of this thesis have been applied.

It is with grateful memories I recall the help and advice that I have received at different phases of doing my Ph.D. work from Director Rein Kuusik and Prof Jaak Tepandi from the Department of Informatics of Tallinn University of Technology.

Thanks to Bob McDonald from Oregon, United States for proofreading my long thesis and checking the English grammar and spelling.

The accomplishing of this thesis would have been impossible without the enormous support that I have received from my wonderful family and parents. My parents have always surrounded me with utmost care and love. In one way or another, they have led me from my early childhood towards a career as a researcher. In 2001 - 2004, I stayed for several months with them while preparing different versions of my thesis.

My wife and children are the best Blessing God could ever give me. They have been so helpful and understanding through this whole long process. While I have worked on my Ph.D. thesis, I also had my everyday work. My wonderful wife and children have had to endure me being away from home almost every Saturday and staying long evenings at work at least two to three evenings per week throughout the last five years. I missed them during this time. This was the most difficult barrier to writing my thesis. The time I had to spend away from my family. In addition, I have also spent the lion's share of my vacations for the past three years writing down my thesis. I am very much indebted to my beloved wife Siiri and wonderful daughters Eliise (5 years) and Sanne (2 years) for all that. I am also thankful to my wife Siiri for the technical help with the editing of the thesis.

And last but not least, I thank our Heavenly Father who has made all this possible, but has also revealed to us through Jesus Christ that there are more important things already in this world than writing Ph.D. theses.

Kuldar Taveter

In Espoo, Finland and Randvere, Estonia, 2001 - 2004.

Preface

The author's interest in business modelling based on business rules took rise in the project of developing a compiler for the COBOL programming language that he participated in more than ten years ago. As is generally known, a large number of business rules used to be (and to some extent is still) embedded in COBOL programs. After business rules have been extracted from computer programs or newly created, the problem is how to represent them in a structured way. To this end, quite a few techniques for representing business rules have emerged. However, only a few of them connect business rules' modelling to the modelling of business processes and even fewer techniques do this in a precise and traceable way. As a part of the integration between business rules and processes, each business rule needs to be attached to a specific actor. Since business rules generally access data, their integration with information and data models is self-evident. Finally, the event part of a business rule, which is structured in the Event-Condition-Action manner, serves as an interaction-related component of the rule. In addition to enabling simulation of business processes, executable business models achieved this way provide an invaluable support for business process automation which is gaining momentum.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. BACKGROUND

1.1.1. Business Rules

According to [Zave97a], requirements engineering must address the contextual goals why software is needed, the functionalities the software has to accomplish to achieve those goals, and the constraints restricting how the software accomplishing those functions is to be designed and implemented. As it can be seen, in the center of requirements are *functional requirements* which are based on organizational goals and determine and restrict the design and implementation of the software. It is claimed in [Gottesdiener99] that the true essence of functional requirements is made up of *business rules* which "exist only to support the goals of the business". This opinion is also supported in [Sandy99], where it is stated that an important reason for the failure of poorly designed systems to satisfy important organizational requirements is a lack of explicit analysis of the organizational rules. Also numerous other sources, like [Loucopoulos91], [Moriarty93], [Herbst97], [BR00], and [Ross03], emphasize the importance of business rules in requirements acquisition.

Historically, most of the research and development work in the area of business rules is connected to active databases. It originated in 1988 when Dayal proposed in [Dayal88] the notion of Event-Condition-Action (ECA) rule in the context of active databases. Later on, this line of research has been continued in [Bussler94], [Halpin96], [Herbst97], [Ross97], [Berndtsson97], and [Kappel98], among others. A few months later than the paper [Dayal88] appeared, Van Assche proposed in [VanAssche88] independently When-If-Then rules which, upon close inspection, are the same as Dayal's ECA rules. The difference between them is that ECA rules are defined from a data point of view, while the perspective of When-If-Then rules is more process-oriented. The research direction of When-If-Then rules has been continued in [Loucopoulos91] and more recently in [Kardasis03]. The rules in approaches of both kinds form elements of a centralized business rules repository and are thus not attached to any organizational agents (actors).

According to [Sandy99], North-American and European views on business rules can be distinguished. The most notable representatives of the North-American view are Halpin [Halpin96], Ross [Ross97], and the GUIDE project [BR00]. Within the North-American view, business rules are defined as constraints upon creation, updating, and removal of persistent data. The European view is best represented by Herbst [Herbst97]. This view structures rules directly according to the Event-Condition-Action technique used in active database research. It has been concluded in [Hurlbut98] that under the North-American view each rule is decomposed into component pieces with accompanying graphical notation, such as constraints and attributes, while the European view treats rules in a behavioural context, such as Event-Condition-Action statements. It is thus an interesting observation that just like at the birth of the notion of business rule, the European view on business rules is still more process-oriented than the North-American one.

In [Bussler94] and [Kappel98], ECA rules associated with an active database were used for agent coordination in a workflow management system. In both works, a business process is characterized by functional, behavioural, informational, and organizational aspects where *agents* (either human or artificial) are the basic elements of an organization. The activities performed by the agents are coordinated and synchronized by centralized ECA rules.

The level of abstraction of the notion of ECA rule is too low to employ it in the process of requirements acquisition, because, according to [Zave97b], there should be nothing in the requirements that could be considered as an implementation bias (like the association of an ECA rule with an active database). To the best of our knowledge, Wagner was the first one who brought ECA rules to a higher level of abstraction by defining in [Wagner96] and [Wagner98] a generalized *reaction rule* as a rule determining the behaviour of an agent (actor) in response to environment events perceived by the agent and to communication events created by communication acts of other agents. A few years later, in a paper by Odell [Odell99], a business rule of a reactive agent was expressed in the WHEN event IF condition(s) THEN action form, and it was claimed that reactive agent behaviour can be described using such business rules.

In [Berndtsson97], a methodology of generating ECA rules from models of high-level speech acts by state diagrams for a specific active database system associated with an agent was proposed. As far as we know, this was to be the first work in the active database community where ECA rules were attached to agents. However, the work described in [Berndtsson97] also had the implementation bias towards active database systems that was mentioned earlier.

1.1.2. Agent-Oriented Information Systems

We claim that business rules can be most naturally described and defined using an agent-oriented approach. Agent is an emerging abstraction that the field of business information systems may also benefit from. ***Agent*** is understood as an *active* entity, possessing the features of *autonomy*, *proactiveness*, *responsiveness*, and *social behaviour* [Jennings98] in contrast to a passive entity meant for representing information – *object*. Agents thus promote *autonomous action and decision-making* which enables *peer-to-peer* interaction, while objects are better suited to the more rigid *client-server* model [Barbuceanu99].

According to [Wagner03a], there are several approaches to defining agents in the literature, only two of them being relevant for the purposes of this thesis:

1. The software engineering approach emphasizes the significance of application-independent high-level agent-to-agent communication as a basis for general software interoperability. E.g., in [Genesereth94], the following definition of agents is proposed: "An entity is a software agent if and only if it communicates correctly in an agent communication language."
2. The mentalistic approach, based on the knowledge representation paradigm of AI, points out that the state of an agent consists of mental components such as beliefs, perceptions, memory, commitments, expectations, goals, and intentions, and its behaviour is the result of the concurrent operation of its perception (or event handling) system, its knowledge system (comprising an update and an inference operation), and its action system (responsible for epistemic, communicative, and physical actions and reactions). E.g., in the approach of [Shoham93], "an agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments."

One of the cornerstones of the emerging paradigm of Agent-Orientation is an ***ontological distinction between agents and objects***, according to which agents are active entities that can perceive events, perform actions, communicate, or make commitments. Ordinary objects are passive entities with no such capacities.

Agent-Orientation offers a range of high-level abstractions that facilitate the conceptual and technical integration of communication and interaction with established information system technology. Agent-Orientation is highly significant for business information systems since business processes are driven by and directed towards agents (or actors[1]), and hence have to comply with the physical and social dynamics of interacting individuals and institutions. Agent-Orientation emphasizes the fundamental role of agents and their mental state, and of communication and interaction, for analyzing and designing organizations and organizational information systems. This turns out to be crucial for a proper understanding of business rules. Since these rules define and constrain the interactions among business agents, they have to refer to the components of their mental state, such as the knowledge/information and the commitments of an organization.

An agent is thus not just a technological building block (*software agent*) like it is sometimes understood [Jennings00], but also an important modelling abstraction that can be used at all logical levels (e.g., according to the ISA framework) in the development of an information system.

As it is stated in the AOIS Glossary included in [AOIS00], ***agent-oriented information systems*** (AOIS) represent a new information system paradigm where communication between different (software-controlled) systems and between systems and humans is understood as communication between agents whose state consists of mental components (such as beliefs, perceptions, memory, goals, commitments, etc.). In enterprise information systems, for instance, the AOIS paradigm implies that business agents are treated as first class citizens along with business objects.

We add to this our own definition stating that an ***agent-oriented information system*** is an information system where the abstraction of an agent is used *at least* at the levels of business and information system models, but possibly also at the levels of the technology model and implementation.

AOIS are closely related to another, yet broader emerging paradigm: the paradigm of ***Cooperative Information Systems*** (CIS), according to which information systems are viewed as consisting of agents who relate to each other as a social organization. Agents cooperate when they share goals and work together to fulfill those goals [DeMichelis97].

For the development and maintenance of agent-oriented information systems, a set of suitable business modelling techniques is required. We term a modelling approach comprising the appropriate

[1] We use the terms 'actor' and 'agent' as synonyms.

modelling techniques ***agent-oriented modelling***. We claim that agent-oriented modelling enables to capture more semantics of the dynamic aspects of business modelling, such as the events and actions related to the ongoing business processes of an enterprise, than traditional approaches like, e.g. UML [OMG03a].

1.2. PROBLEM STATEMENT

In our work, we avoid the implementation bias mentioned in section 1.1.1 by attaching business rules to abstract actors (agents) which are not associated with any software systems. However, this is just the first step towards realizing the full potential of business rules in business modelling which has not been achieved prior to this work. In our opinion, the real promise of agent-oriented modelling lies in the creation of *executable models*. Such models can be used for simulation of business and/or manufacturing processes. Moreover, the transition from executable models of business and/or manufacturing processes to the specifications of the software that partly or fully automates theses processes is straightforward. This is especially true in cases where automation is accomplished through the use of artificial (e.g. software) agents, even though an agent-oriented business modelling techniques should not impose it in any way.

This presents a challenge to create a modelling notation and methodology that (1) could be used for the creation of business models of different perspectives like of the informational, interact ional, and behavioural perspectives and that (2) could be used at all stages of business modelling like analysis and design and their substages. The modelling steps of such a methodology would lead to executable business process models that could serve as a basis for business or manufacturing process automation using appropriate software solutions. Hopefully this thesis manages to create the first version of such a methodology.

1.3. OUTLINE OF THE THESIS

In this Chapter 1 of the thesis, which presents the background of the study, we first provide our own definitions and classifications of business rules and processes and present an overview of the existing business modelling frameworks. Thereafter we argue for the need to introduce views (perspectives) of agent-oriented modelling, propose such views, and discuss the position of business rules in agent-oriented modelling. We conclude Chapter 1 by setting the research objectives, determining the scope of the research, and describing the research approach.

In Chapter 2, the literature review is presented in the form of comparative evaluation of five business modelling languages and notations for business modelling and three business modelling methodologies with respect to the six views of agent-oriented modelling. As a result of the comparison, the need for a distinctive multi-perspective technique and methodology of agent-oriented modelling is argued for. Other related work is also briefly described in Chapter 2.

In Chapter 3, we first present the running example of car rental to be used throughout the chapter. We then introduce a distinction between the Agent Layer and Object Layer of business modelling. After that, we define the metamodel of the Business Agents' Approach and propose to use for agent-oriented modelling based on it a combination of the Agent-Object-Relationship Modelling Language (AORML) and the Object Constraint Language (OCL) of UML. We also briefly describe both AORML and OCL. Next, we extend AORML by activity diagrams by relating them to the notions of AORML and OCL and explaining their functioning. We also provide an operational semantics for activity diagrams and define a semi-formal activity modelling language equivalent to activity diagrams. Thereafter we propose a multi-perspective modelling methodology and process that are based on the metamodel of the Business Agents' Approach. We describe the modelling under each view of agent-oriented modelling by using the running example of car rental. Finally, we describe how activity diagrams can be "executed" on the JADE agent platform.

In Chapter 4, the methodology worked out is applied to the case studies of the ceramic factory and advertising. The purposes of the first case study are simulation of the factory and preparing for the creation of a semiautomatic manufacturing control system for the factory. The second case study is aimed at automation of inter-enterprise business processes related to advertising. For both case studies, the applying of the methodology under each view is described.

In the final chapter, Chapter 5, we present the summary and an overview of the main contributions of the thesis. We also discussion the results of the study and describe both the ongoing research work and the proposals for future research work.

Appendix A defines the grammar for the modification of OCL proposed in the thesis. Appendix B presents the EBNF grammar of the activity modelling language that is introduced in Chapter 3. Appendixes C and D define derivation rules for the case studies of car rental and ceramic factory, respectively. Appendix E consists of those extended AORML activity diagrams of the case study of car rental that are not included by Chapter 3. Appendixes F and G present activity diagrams of the extended AORML for the case studies of ceramic factory and advertising. Finally, Appendix H lists the publications where some results reported in this dissertation have appeared.

1.4. BUSINESS RULES AND PROCESSES

Our definitions of business rules and processes and classifications of business rules given below were first presented in [Taveter01c].

1.4.1. Definitions and Classification of Business Rules

The term business rule can be understood both at the level of a business domain and at the operational level of an information system. The more fundamental concepts are business rules at the level of a business domain. In certain cases, they can be automated by implementing them in an information system, preferably in the form of an executable specification. It should be the goal of advanced information system technology to provide more support for business rules in the form of high-level machine-executable declarative specifications, similar to the SQL concepts of assertions and triggers.

1.4.1.1. Business Rules at the Business Level

At the business level, business rules are defined in the literature as

- statements describing something affecting the enterprise that limits the actions that can be taken [Bubenko93];
- statements that define or constrain some aspect of the business [BR00];
- statements about how the business is done, i.e., about guidelines and restrictions with respect to states and processes in an organization [Herbst97];
- laws or customs that guide the behaviour or actions of the actors connected to the organization [VanAssche88];
- declarations of policy or conditions that must be satisfied [OMG92].

As pointed out in [Gottesdiener99], business rules are at the core of functional requirements. As the essential ingredient of functional requirements, business rules deserve direct, explicit attention. When the rules are not explicit, and if developers encode them by guessing, the essential business rules may be discovered as missing or wrong during latter phases.

Business rules can be enforced on the business from the outside environment by regulations or laws, or they can be defined within the business to achieve the goals of the business. A business rule is based on a *business policy*. An example of a business policy in a car rental company is ”only cars in legal, roadworthy condition can be rented to customers” [BR00]. According to [Martin98], business rules allow user experts to specify policies in *small*, *standalone units* using explicit statements. Business rules are *declarative* statements: they describe *what* has to be done or *what* has to hold, but not *how*.

We define business rules as follows: ***Business rules are statements that express*** (certain parts of) ***a business policy, such as defining business terms, defining deontic assignments (of powers, rights and duties), and defining or constraining the operations of an enterprise, in a declarative manner*** (not describing/prescribing every detail of their implementation).

According to [BR00] and [MDC99], business rules can be divided into ‘structural assertions’ (or ‘term rules’ and ‘fact rules’), ‘action rules’, and ‘derivation rules’.[2] Similarly, Bubenko et al [Bubenko01] categorize business rules into ‘constraint rules’, ‘event-action rules’, and ‘derivation rules’, while Martin and Odell [Martin98] group rules into two broad classes, ‘constraint rules’ and ‘derivation rules’ (remarkably, they subsume ‘stimulus response rules’ – which we call *reaction rules* – under ‘constraint rules’). [Herbst97] distinguishes between ‘integrity rules’ (that are further divided into static and dynamic integrity constraints) and ‘automation rules’.

In [BR00], a further class of business rules, called ‘authorizations’, is proposed. They represent a particular type of *deontic assignments*. However, the term ‘authorization’ is ambiguous. In many cases, it is synonymous to *right* (and *permission*). But in some cases it rather denotes an *institutional power*. Rights define the privileges of an agent (type) with respect to certain (types of) actions. Complementary to rights, we also consider *duties*.

In summary, three basic types of business rules have been identified in the literature: *integrity constraints* (also called ‘constraint rules’ or ‘integrity rules’), *derivation rules*, and *reaction rules* (also

[2] ‘Structural assertions’ introduce the definitions of business entities and describe the connections between them. Since they can be captured by a conceptual model of the problem domain, e.g. by an Entity-Relationship (ER) or a UML class model, we do not consider them as business rules but rather as forming the business *vocabulary* (or *ontology*).

called ‘stimulus response rules’, ‘action rules’, ‘event-action rules’, or ‘automation rules’). A fourth type, *deontic assignments*, has only been partially identified (in the proposal of considering ‘authorizations’ as business rules).

An ***integrity constraint*** is an assertion that must be satisfied in all evolving states and state transition histories of an enterprise viewed as a discrete dynamic system. There are state constraints and process constraints. *State constraints* must hold at any point in time. An example of a state constraint is: “a customer of the car rental company EU-Rent must be at least 25 years old”. *Process constraints* refer to the dynamic integrity of a system; they restrict the admissible transitions from one state of the system to another. A process constraint may, for example, declare that the admissible state changes of a RentalOrder object are defined by the following transition path: isReserved → isAllocated → isEffective → isDroppedOff.

A ***derivation rule*** is a statement of knowledge that is derived from other knowledge by an inference or a mathematical calculation. Derivation rules capture terminological and heuristic domain knowledge that need not to be stored explicitly because it can be derived from existing or other derived information on demand. An example of a derivation rules is: “the rental rate of a rental is inferred from the rental rate of the group of the car assigned to the rental”.

Reaction rules are concerned with the invocation and sequencing of actions in response to events. They state the conditions under which actions must be taken; this includes triggering event conditions, pre-conditions, and post-conditions (effects). An example of a reaction rule from the domain of car rental is: “when receiving from a customer a request to reserve a car of some specified car group, the branch checks with the headquarters to make sure that the customer is not blacklisted”.

The triggering event conditions in the definitions of reaction rules in [BR00], [Herbst97], [Bubenko01], and [MDC99] are either explicitly or implicitly bound to update events in databases. Depending on some condition on the database state, they may lead to an update action and to system-specific procedure calls. In contrast to this, we choose the more general concept of a reaction rule as proposed in [Wagner98]. Reaction rules define the behaviour of an agent in response to environment events (perceived by the agent), and to communication events (created by communication acts of other agents).

Deontic assignments of powers, rights and duties to (types of) internal agents define the deontic structure of an organization, guiding and constraining the actions of internal agents. An example of a deontic assignment statement is: “only the branch manager has the right to grant special discounts to customers”.

1.4.1.2. Business Rules and Goals

Business rules may also serve to operationalize business goals. In the EKD framework [Bubenko01] which is based on the enterprise model [Bubenko93] [Bubenko94], the *Goals Model* focuses on describing the goals of the enterprise. It states what the enterprise and its employees want to achieve, or to avoid, and when. The Goals Models usually clarify questions, such as: where should the organization be moving, what are the goals of importance for the organization, criticality, and priorities of these goals, how are goals related to each other, and which problems are hindering achievement of goals. The *Business Rule Model* of EKD is used to define and maintain explicitly formulated business rules, consistent with the Goals Model. Business rules may be seen as operationalisations or limits of goals. Business rules may be in the form of:

- precise statements that describe the way that the business has chosen to achieve its goals and to implement its policies or,
- the various externally imposed rules on the business, such as regulations and laws.

The Business Rules Group offers even a more refined operationalization of business goals into business rules in its Business Rule Motivation Model [OBP00]. There are two major areas of the Business Rule Motivation Model:

- The first is the *ends* and *means* of business plans. Among the ends are things the enterprise wishes to achieve — for example, *goals* and *objectives*. Among the means are things the enterprise will employ to achieve those ends — for example, *strategies*, *tactics*, *business policies*, and *business rules*.
- The second is the *influences* that shape the elements of the business plans, and the *assessments* made about the impacts of such influences on ends and means (i.e., *strengths*, *weaknesses*, *opportunities*, and *threats*).

A ***business rule*** within this context is a directive, intended to influence or guide business behaviour, in support of a business policy that has been formulated in response to an opportunity, threat, strength, or weakness.

1.4.1.3. Business Rules at the Level of an Information System

In certain cases, business rules expressed at the business level can be automated by mapping them to executable code at the information system level as shown in Table 1-1. This mapping is, however, not one-to-one, since programming languages and database management systems offer only limited support for it. While general purpose programming languages do not support any of the three types of expressions (with the exception of the object-oriented language Eiffel that supports integrity constraints in the form of 'invariants' for object classes), SQL has some built-in support for constraints, derivation rules (views), and limited forms of reaction rules (triggers).

Table 1-1. Mapping of business rules from the business level to the information system level using currently available technology.

Concept	***Implementation***
Constraints	if-then statements in programming languages; DOMAIN, CHECK, and CONSTRAINT clauses in SQL table definitions; CREATE ASSERTION statements in SQL database schema definitions
Derivation Rules	deductive database (or Prolog) rules; SQL CREATE VIEW statements
Reaction Rules	if-then statements in programming languages; CREATE TRIGGER statements in SQL; production rules in 'expert systems'
Permission Rules	role-based access rights
Prohibition Rules	(not available)
Duty Assignments	(not available)
Empowerment Rules	(not available)

1.4.2. Business Processes

Business rules define and control *business processes*. A widely accepted definition of a business process is [Davenport93]: "A business process can be defined as a collection of activities that takes one or more kinds of input, and creates an output that is of value to the customer". In [Hammer93] this definition is paraphrased by stating: "A [business] process is simply a structured set of activities designed to produce a specified output for a particular customer or market". A business process describes from start to finish the sequence of events required to produce the product or service [Yourdon96]. These definitions are criticized in [Smith03] as not capable ones of explaining the true nature of collaborative and transactional business processes of today. The following new definition is proposed in their place: "A business process is the complete and dynamically coordinated set of collaborative and transactional activities that deliver value to customers". Business processes typically involve several different functional organization units. Often business processes also cross organizational boundaries.

We prefer to adopt a more general perspective and consider a business process as a special kind of a *social interaction process*. Unlike physical or chemical processes, social interaction processes are based on communication acts that may create commitments and are governed by norms. We distinguish between an interaction process type and a concrete interaction process (instance), while in the literature the term 'business process' is ambiguously used both at the type and the instance level.

We thus refine and extend the definitions of [Smith03], [Yourdon96], [Hammer93], and [Davenport93]: ***a business process is a social interaction process for the purpose of doing business***. We view ***a social interaction process*** as a temporally ordered, coherent set of dynamically coordinated events and actions, involving one or more communication acts, perceived and performed by agents, and following a set of rules, or protocol, that is governed by norms, and that specifies the

type of the interaction process. Notice that we did not choose *activities* as the basic elements of a process. While an ***action*** happens at a time point (i.e., it is immediate), an ***activity*** is being performed during a time interval (i.e., it has duration), and consists of a set of actions. However, in section 3.6 we will provide modelling constructs for the modelling of both actions and activities.

1.5. BUSINESS MODELLING FRAMEWORKS

1.5.1. Perspectives on Business Modelling

Traditional business modelling can be described by the so-called "triangle" model [Nilsson98]. The "triangle" consists of three interrelated components: *data model* of the problem domain, relevant *events* taking place in the domain, and *functions* of the domain that are triggered by the corresponding events and operate on the instances of entity types (object classes) of the data model (v. Figure 1-1). As it can be seen from Figure 1-1, objects, processes, and transactions can be represented as combinations of data and functions, functions and events, and events and data, respectively.

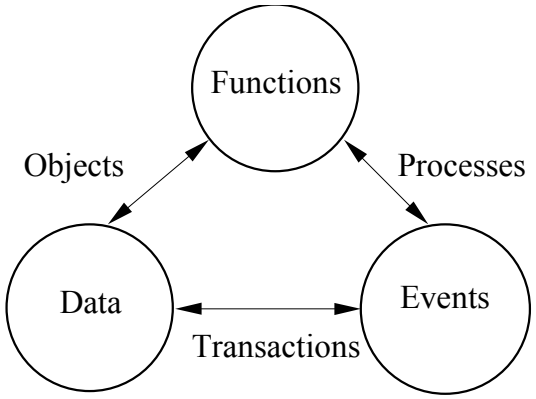


Figure 1-1. The "triangle" model.

1.5.2. The Information Systems Architecture (ISA) Framework

According to [Zachman87] and [Sowa92], different aspects that should be addressed in business modelling and information systems development are **data (what?)**, **function (how?)**, **network (where?)**, **actors (who?)**, **time (when?)**, and **motivation (why?)**. The aspects mentioned are represented as the six columns of the framework for information systems architecture (*ISA framework*) [Zachman87] presented in Table 1-2. The ISA framework was originally proposed in [Zachman87] and extended in [Sowa92].

The *data (or concepts) aspect* describes things important to the business. It clarifies what concepts or subjects the business is about and how are they defined. Each of the concepts has one or more relationships that link it to other concepts. The representation of all the concepts and their relationships to other concepts constitutes the total *data* of the working system [Sowa00].

The *function aspect* describes the activities and processes performed within the business. Each activity takes one or more concept types as arguments [Sowa00]. A business process can be defined as a collection of activities that takes one or more kinds of input, and creates an output that is of value to the customer [Hammer93]. A business process describes from start to finish the sequence of events required to produce the product or service [Yourdon96].

Table 1-2. The Information Systems Architecture (ISA) framework

	Data (What?)	Function (How?)	Network (Where?)	Actors (Who?)	Time (When?)	Motivation (Why?)
Scope	List of things important to the business	List of processes the business performs	List of locations in which the business operates	List of organization units of the business	List of events significant to the business	List of business goals and strategies
Model of the Business	ER-diagram (including m:m, n-ary attributed relation-ships)	Business process model (process flow diagram)	Logistics network (nodes and links)	Organization chart with roles, skill sets, and authori-zations	Business master schedule	Business plan with objectives and strategies
Model of the Infor-mation System	Data model (1:m relation-ships, fully normalized)	Data flow diagram; application architecture	Distributed system architecture	Human interface architecture (roles, data, access)	Depen-dency diagram, entity life history	Business rules' model
Techno-logy Model	Data architecture (tables and columns); mapping to legacy data	System design: structure chart, pseudo-code	System architecture (hardware, software types)	User interface (how the system will behave); security design	"Control flow" diagram (control structure)	Business rules' design
Compo-nents	Physical data storage design	Detailed program design	Network architecture and protocols	Screens, security architecture (who can see what?)	Timing model	Specification of business rules in program logic
Func-tioning System	Converted data	Executable programs	Com-munication facilities	Trained people	Business events	Enforced business rules

The *network aspect* is concerned with the geographical distribution of the activities of the business. At the most general level, it is simply a list of locations in which the business operates. At a lower level of abstraction, it becomes a more detailed communications chart, describing how the various locations interact with each other [Hay97]. Each location has one or more links that connect it to other locations [Sowa00].

The *actors aspect* describes who is performing which processes and activities. This aspect has to do with the allocation of work and the structure of authority and responsibility [Sowa92], i.e. with the design of the organization. The actors include humans, such as employees and customers, and computerized agents that operate automatically. Each actor has associated activities, tasks, or work that he/she/it performs [Sowa00].

The *time aspect* describes events significant to the business. Here time is abstracted out of the real world to design the event-to-event relationships that establish the performance criteria and quantitative levels of enterprise resources [Sowa92]. Each event occurs on some cycle, which may be periodic, such as a billing cycle, or irregular, such as demand-driven events initiated by various actors. The totality of events and cycles determines the *schedule* [Sowa00].

The time aspect also includes coordination relationships between different events performed by different actors as described e.g. in [Singh00]. Since it is difficult to address this aspect in isolation from the others, especially from the function aspect, many business modelling methodologies, including Eriksson-Penker extensions to UML [Eriksson99] and EKD [Bubenko01], combine the time aspect with the function aspect.

The *motivation aspect* describes the goals of the enterprise. Goals can be decomposed into sub goals and allocated to individual actors, activities, or processes. The motivation aspect also concerns the translation of business goals into specific ends. Each end has an associated means by which it may be accomplished [Sowa00]. The means is expressed as one or more explicitly formulated business rules. The totality of all ends and means constitutes the strategy [Sowa00].

Some business modelling methodologies, such as EKD [Bubenko01] and Eriksson-Penker extensions to UML [Eriksson00], divide the motivation aspect into the models of goals and business rules.

The aspects described can be viewed from different perspectives at different logical levels. These logical levels together with their corresponding perspectives are represented as rows of Table 1-2. According to Zachman [Zachman87], the six logical levels in the development of an information system and the corresponding perspectives (given in parenthesis) are:

- **Scope** (the perspective of *planner*). It corresponds to an executive summary for a planner or investor who wants an estimate of the scope of the information system, its purpose, what it would cost, and how it would perform.
- **Enterprise or business model** (*owner*). Constitutes the design of the business and shows the business entities and processes and how they interact. The entities at the enterprise level are the actors, resources, products, and tasks of the business [Sowa00].
- **System model** (*designer*). Describes the information system as designed by a systems analyst who must determine the data elements and functions that represent business entities and processes.
- **Technology model** (*builder*). A model that must adapt the information system model to the details of the programming languages, input/output devices, or other technology.
- **Models of components** (*subcontractor*). Detailed specifications that are given to programmers who code individual modules without being concerned with the overall context, purpose, or structure of the system.
- **Functioning system** (*user*). The view of the completed information system that is made part of an organization.

1.5.3. The Enterprise Model

The enterprise model was first proposed in [Bubenko93] and refined in [Bubenko94]. According to [Bubenko93], the enterprise model has to answer the following questions:

- Why is the information system built? What is its justification?
- Which are the business processes, and which of these are to be supported by the information system?
- Which are the actors of the organization performing the processes?
- What concepts are they processing or talking about; which are their information needs?
- Which initial objectives and requirements can be stated regarding the information system to be developed?

The enterprise model includes the following interrelated sub models, as shown in Figure 1-2, adopted from [Bubenko94]:

- **The objectives sub model**. It is intended for describing and discussing the reason or motivation for activities, actors, and concepts of the other sub models – it addresses the "why"-perspective of the enterprise and development of the information system for it [Bubenko93]. Goals and business rules for a particular enterprise activity (or a set of activities), existing, to be modified, or to be designed, are stated, and their relationships analyzed [Bubenko94].
- **The concepts sub model**. The concepts sub model is used to define the "ontology" of the "universe of discourse" of interest, i.e. the set of object types, relationships, and object properties of the problem domain we are talking about. In this sub model business rules of the objectives model are also further refined into static as well as dynamic rules for the states of the concepts sub model as well as for permissible state changes [Bubenko94].
- **The actors sub model**. This sub model is used to discuss and define the set of actors of each studied activity (individuals, groups, job-roles/positions, organizational units, machines, etc.), and their inter-relationships, such as part-of, reports-to, etc. [Bubenko94].
- **The activities and usage sub model.** In this part of a requirements specification, the particular organizational activity (in a wide sense), existing, to be modified, or to be developed, is defined

and described from the point of view of activities, tasks, processes, and the information and material flows between them [Bubenko94].

Information system requirements, related to the above models, are described by the following two sub models [Bubenko94]:

- **The functional requirements sub model**. This part of the requirements specification elaborates the specific objectives and requirements that are put on the information system to support the activities and objectives of the enterprise listed previously.
- **The non-functional (NF) requirements sub model**. NF requirements can be of several different kinds, such as standards for interfacing other systems, restrictions concerning the use of hardware and/or software, accuracy of information, timeliness of information, security, access rights, economic limits of the development project, etc. NF requirements are primarily related to the activity and usage sub model, and indirectly to the objectives sub model, as activities are normally motivated by the objectives sub model.
- The information system is described conceptually by the **information system's sub-model**. By this we mean the complete, formal specification of the information system such that designated activities and processes in the activities and usage sub-model, as well as the functional and NF requirements are supported (to a higher or lesser degree).

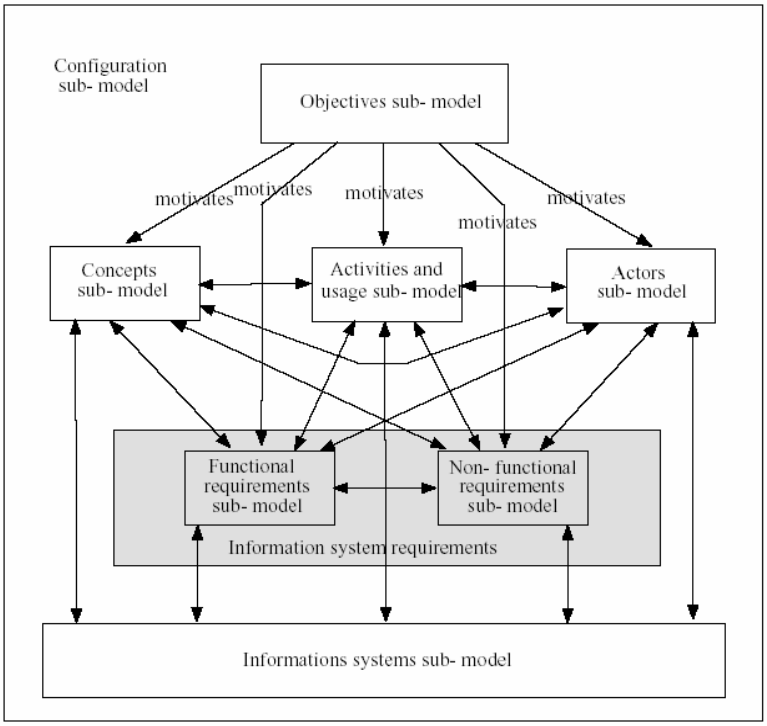


Figure 1-2. The enterprise model and its interrelated submodels

1.5.4. Conceptual Framework for Process Modelling by Curtis et al

Separately from the ISA framework and the enterprise model, Curtis et al [Curtis92] defined a conceptual framework for process modelling. They claim that the constructs that collectively form the essential basis of a process model are:

- **Agent** – an actor (human or machine) who performs a process element.
- **Role** – a coherent set of process elements to be assigned to an agent as a unit of functional responsibility.
- **Artifact** – a product created or modified by the enactment of a process element.

The authors state in [Curtis92] that "among the forms of information that people ordinarily want to extract from a process model are what is going to be done, who is going to do it, when and where will it be done, how and why will it be done, and who is dependent on its being done". They define the following most commonly represented perspectives:

- **Functional perspective** represents what process elements are being performed, and what flows of informational entities (e.g., data, artifacts, products) are relevant to these process elements.
- **Behavioural perspective** represents when process elements are performed (e.g., sequencing), as well as aspects of how they are performed through feedback loops, iteration, complex decision-making conditions, entry and exit criteria, and so forth.
- **Organizational perspective** represents where and by whom (which agents) in the organization process elements are performed, the physical communication mechanisms used for transfer of entities, and the physical media and locations used for storing entities.
- **Informational perspective** represents the informational entities produced or manipulated by a process; these entities include data, artifacts, intermediate and end products, and objects; this perspective includes both the structure of informational entities and the relationships among them.

According to [Curtis92], these perspectives underlie separate yet interrelated representations for analyzing and presenting process information.

Curtis et al also analyze the applicability of different language types and modelling approaches for process modelling. The results of their survey are given in Table 1-3.

Table 1-3. Applicability of different language types and modelling approaches for process modelling

	Perspectives			
	Functional	**Behavioural**	**Organizational**	**Informational**
Procedural programming languages	+	+		+
Systems analysis and design	+		+	+
AI languages and approaches	+	+		
Events and triggers		+		
State transitions and Petri nets	+	+	+	
Control flow		+		
Functional languages	+			
Formal languages	+			
Data modelling				+
Object modelling		+[3]	+	+
Precedence networks		+		

1.5.5. Evaluation and Comparison of the Frameworks

The main shortcoming of the "triangle"-based business modelling, introduced in Section 1.5.1, including object-oriented modelling, is that it doesn't explicitly deal with *actors* that perform different business functions. For example, the data flow diagram, which is one of the most popular ways of modelling the vertex of functions of the "triangle" model, views functions as transformers of data flows that are not attached to any actors performing these functions. True, on workflow diagrams functions are attached to actors, but these actors are not included by the business model.

Actors are often represented as instances of entity types of the data model, but there they are *passive* entities to be manipulated with rather than *active* performers of business functions. This is reflected by the existing methodologies of modelling and designing object-oriented systems like e.g. UML [OMG03a]. In the UML, the customers and the employees of a company would have to be modelled as 'objects' in the same way as rental cars and bank accounts.

It has also been noticed by the others [Wagner99] that in the UML actors are only considered as users of the system's services in "use cases", but otherwise remain external to the system model. Due to this business rules defining and constraining the functions of the business remain up in the "air" and are not attached to any processors/executors.

[3] The behavioural perspective covered by object modelling (object life history) must have forgotten in [Curtis92].

Table 1-4. Comparison of frameworks for conceptual modelling

		ISA frame-work					
		Data	**Function**	**Network**	**Actors**	**Time**	**Motivation**
Enterprise model	**Objectives**						+
	Concepts	+					
	Actors			+	+		
	Activities and usage	+	+			+	
Conceptual framework by Curtis et al	**Functional**		+				
	Behavioural		+			+	
	Organizational			+	+		
	Informational	+					

Organizational perspective and actors are present in the three other frameworks reviewed. In Table 1-4 the enterprise model and the conceptual framework for process modelling by Curtis et al are compared to the ISA framework, as to ”a richest theoretical framework in use” [Kirikova00]. As Table 1-4 shows, the sub models of the enterprise model can be more or less precisely mapped to the ISA framework. However, the enterprise model does not explicitly include a sub model that would correspond to the time aspect of the ISA framework. Therefore the mapping of the activities and usage sub model of the enterprise model to the time aspect of the ISA framework is guessed by the author of this thesis.

The functional, behavioural, and informational perspectives of the framework by Curtis et al respectively correspond to the function, time, and data aspects of the ISA framework. The behavioural perspective also includes some constructs that traditionally belong to the function aspect of the ISA framework like feedback loops, iteration, complex decision-making conditions, and entry and exit criteria.

The organizational perspective of the framework by Curtis et al is covered by the network and actors aspects. The motivation aspect of the ISA framework does not have a counterpart in the framework by Curtis et al.

1.5.6. Views of Agent-Oriented Modelling

Any of the three frameworks for conceptual modelling compared in Table 1-4 can serve as a background framework of agent-oriented modelling, because all of them have an important place for actors. Moreover, one of the clear advantages of the ISA framework and the enterprise model that makes them especially suitable for such a purpose is that they do not prescribe a particular model for reflection of the requirements specification [Kirikova00]. The same also applies to the conceptual framework by Curtis et al. However, since the modelling of interactions in the frameworks mentioned is divided between different aspects / sub models / perspectives, we have identified on the basis of these frameworks *six views of agent-oriented business modelling*:

- **Informational view**, concerns the modelling of *passive*, *informational entities*, i.e. objects and relationships between them.
- **Functional view**, concerns the modelling of *activities* the agents are engaged in.
- **Behavioural view**, concerns the modelling of the *order* in which the activities are to be performed, as well as of the agents' *reactions* to the events perceived by them.
- **Organizational view**, concerns the modelling of *active entities*, i.e. agents and agent types.

- **Interactional view**, concerns the modelling of *interactions* and *communication* between the agents.
- **Motivational view**, concerns the modelling of the *goals* attached to the activities that the agents are trying to achieve.

We will follow the six views that were presented in our modelling methodology to be described in Chapter 3.

1.5.7. Position of Business Rules in Agent-Oriented Modelling

As it can be seen in Figure 1-2, the motivation aspect / the objectives sub model[4], that the motivational view defined in section 1.5.6 is based on, serves as the glue that connects all the other aspects / sub models. At the higher level of abstraction, the motivational view describes the goals of the enterprise. Goals are decomposed into sub-goals and allocated to individual actors, activities, and processes. At the lower level of abstraction, as we saw in section 1.4.1.2, each goal is expressed as a combination of one or more business rules.

Business rules of the type *integrity constraints* clearly belong to the informational view, as they have to do with the states of the body of concepts at any point in time and restrict the admissible transitions from one state of the body of concepts to another.

Since *derivation rules* derive a new knowledge from existing knowledge that lies within the informational view, derivation rules also belong to the informational view.

Business rules of the type *deontic assignments* belong to the organizational view because they determine agents' rights and duties to perform the actions that activities consists of.

Reaction rules have a direct connection to the interactional, informational, functional, behavioural, and organizational views of agent-oriented modelling.

Firstly, reaction rules are means of responding to various business *events* which occur during *interactions* between agents. Events can be internal or external in relation to the enterprise.

Secondly, reaction rules access the body of concepts, possibly by making use of derivation rules, and determine necessary state transitions of the body of concepts consisting of *informational entities*.

Thirdly, reaction rules start activities and processes, i.e. *functions* of the enterprise, and control their *behaviour*.

Since activities and processes are driven by and directed towards *actors*, a business rule is always attached to some human or automated actor.

And last but not least, activities are associated with the *goals* that the corresponding actors are trying to achieve.

Consequently, business rules span all six views of agent-oriented modelling. Since reaction rules are directly related to five out of six views of agent-oriented modelling, they seem to be the most important type of business rules. At the same time, *reactive behaviour of agents is the most dominant one in a business domain*. We can thus conclude that reaction rules are those business rules where an agent-oriented approach is most promising.

1.6. RESEARCH OBJECTIVE

Table 1-4 reveals that no one of the different language types and modelling approaches examined there covers more than three modelling perspectives proposed in [Curtis92]. It is additionally stated in [Curtis92] that "an approach that integrates multiple representational paradigms is currently considered necessary for effective software process modelling[5]".

This finding is also supported by the claim in [Nilsson98], according to which no one of the business modelling methodologies available covers equally all three vertexes of the triangle in Figure 1-1.

The approach that has been taken to meet the need expressed above in, for example, UML [OMG03a] is loose integration of various modelling perspectives and paradigms. However, this is not always satisfactory, as the modeler is forced to use different modelling techniques in parallel which is confusing. To be more specific, e.g. in [Lubell02] it is stated with regard to manufacturing or business process modelling: "Unfortunately for process modelers, no single type of UML diagram captures all of the information needed to describe a process. UML activity diagrams do a good job modelling

[4] The corresponding perspective is not present in the conceptual framework for process modelling by Curtis et al.
[5] The paper [Curtis92] is specifically about software development processes but it can also be applied to business processes.

complicated sequences and parallelism. However, activity diagrams are not the best choice for representing the relationships between activities and objects. UML interaction diagrams do a much better job describing how actions and objects collaborate."

Some development in the ability of modelling tools to support different kinds of models has been noticed in [Gates03]: "There is a common theme that can be seen across all new data modelling tools – more focus on business process modelling".

With this background, there seems to be a need for the modelling approach that would simultaneously provide support for different views of agent-oriented modelling defined in section 1.5.6. Therefore a natural research objective for this thesis is to ***work out and apply a modelling notation and methodology that would conform to the following requirements***:

- enables to create and integrate business models of different perspectives;
- can be used at the analysis and design stages of business modelling;
- lends itself to the creation of executable business process models.

Since, as we showed in section 1.5.7, reaction rules span all six views of agent-oriented modelling, it seems natural to associate the methodology and modelling notation to be created with reaction rules. This is compatible with our definition of business processes that was presented in section 1.4.2 where we defined a business process as a social interaction process that is specified and controlled by business rules. Consequently, business processes as well as business rules span all six views of agent-oriented modelling.

Another objective of this dissertation springs from the need expressed in [DeMichelis97] and [AOIS00] for a systematic approach to the development of agent-oriented and cooperative information systems in terms of requirements acquisition, design, and implementation. For this reason, the second objective of this thesis is to ***propose a systematic approach to the development of AOIS and CIS***.

According to [DeMichelis97], in a cooperative information system, once captured organizational objectives and systems requirements must also be "kept alive" and remain a part of the running system, due to ongoing evolution of the system. An adequate objectives' and requirements' representation language should support a *declarative* style of specification which offers the possibility to model requirements adopting an *aerial view* perspective. For example: "the borrowing of a book should be followed by its return within the next three weeks" [DeMichelis97]. In our opinion, representing organizational objectives and systems requirements as *business rules* that define and constrain actions of *business agents* is a step towards this kind of language.

1.7. RESEARCH SCOPE

The logical levels in business modelling and information systems development are best formulated in [Zachman87]. The same levels also apply to agent-oriented modelling. Since this thesis aims at technology-independent analysis and design of information systems, the level addressed by the agent-oriented business modelling methodology proposed in the thesis is the level of **enterprise or business model** (the perspective of *owner*) which was briefly described in section 1.5.2. We divide this level into two sublevels: ***analysis***, where actors (agents) of the problem domain are sketched and each actor is defined in terms of services it provides to other actors, and ***design*** where the focus agent(s) that an information system is to be created for are described within their environment, consisting of other agents. We understand the term 'design' here in the sense of designing a ***socio-technical system***, i.e., a system composed of technical and social subsystems [Pernice95], rather than a purely technical system. At the level of designing a socio-technical system addressed in this work, *we aim to model the functioning and interactions of institutional actors in a precise and executable way, which lends itself to simulation, without necessarily distinguishing between the tasks that are performed by human and automated agents*. This conforms to our understanding of a business process as of a *social interaction process* stated in section 1.4.2.

In accordance with [Wagner03a], at the level of enterprise or business model we adopt the perspective of an external observer who is observing the (prototypical) agents and their interactions in the problem domain under consideration. At the level of *information system model*, which is addressed by the third row of the ISA framework represented in Table 1-2, we adopt the internal (first-person) view of a particular agent to be modelled. However, as was pointed out above, the level of information system model is not treated in this work.

Since modelling of business goals and transforming them into business rules is a wide topic in its own right, we have limited our research objective to the modelling of business rules and business

processes. ***We thus assume that each business rule is already justified by some business goal***. The modelling of business goals has, for example, been treated in [Yu95a], [OBP00], and [Bubenko01]. The present work is confined to the modelling of the goals that are attached to the activities performed by individual agents.

Out of the four types of business rules that were defined in section 1.4.1.1, ***we concentrate on reaction rules***, which were in section 1.5.7 deemed to form the most important type of business rules, ***and derivation rules accessed by them***, and treat the modelling by using integrity constraints just marginally.

1.8. RESEARCH APPROACH

The research approach has been from particular to more general. We have been developing our modelling technique by applying it to the case study of car rental which will be described in section 3.1. For example, at the earlier stage of our work described in [Taveter02a], we have employed ***i****, which is described in section 2.1.4, at the analysis step of the proposed modelling methodology. Later on, we have abandoned this approach in favor of goal-based use cases because of the obvious complexity of the ***i****-to-AOR conversion for an ordinary user.

2. COMPARATIVE EVALUATION OF BUSINESS MODELLING TECHNIQUES

From among a large number of modelling languages, notations, and methodologies for business modelling, we selected for overview and comparative evaluation the techniques in which agents/actors and/or business rules of the types defined in section 1.4.1.1 explicitly or implicitly play a prominent role.

2.1. MODELLING LANGUAGES AND NOTATIONS FOR BUSINESS MODELLING

2.1.1. Ross Notation and Proteus

The Ross Notation [Ross97] is proposed for formalizing and visualizing constraints, conditions, and derivation rules. A *business rule* is understood in [Ross97] as "a constraint or a test exercised for the purpose of maintaining the integrity (i.e., correctness) of data". According to [Ross97], the purpose of a rule generally is to control the updating of persistent (i.e., stored) data – in other words, the results that the execution of actions (processes) are permitted to leave behind. A rule embodies a *business rule statement* which is a formal, implementable expression of some "user requirement", usually stated in textual form using a natural language (e.g., structured English).

Within the Ross Notation, *events* are understood as update events in a database. Every rule can be decomposed into two or more update events. A business rule, however, usually need not – and should not – make any reference to these update events or triggers.

2.1.1.1. Classification of Rules

According to [Ross03], the Ross Notation divides business rules into the following basic rule types:

- *Rejector*: any rule that tends to disallow (that is, reject) an event if a violation of the rule would result.
- *Projector*: any rule that tends to take some action (other than rejection) when a relevant event occurs.
- *Producer*: any rule that neither rejects nor projects events but simply computes or derives a value based on some mathematical function(s).

Two varieties of rules are distinguished between in [Ross97]:

- An *integrity constraint* is a rule that must always yield true (or unvalued). It has enforcement power because it never is permitted to yield false.
- A *condition* is a rule that may yield either true or false (unvalued). Since it is permitted to yield false, it lacks direct enforcement power. Its usefulness arises in providing a test for the enforcement (or testing) of one or more other rules that are enforced (or tested) only while the condition yields true.

Integrity constraints and conditions are given distinct graphic symbols. According to the Ross Notation, each business rule consists of an anchor, rule symbol, and correspondent. *Anchor* is a data type or another rule for whose instances a rule is specified. In the graphical representation of the Ross Notation, the anchor connection *exits* the anchor and *enters* the rule symbol. *Correspondent* is a data type, another rule, or action whose instances are subject to the test exercised by the rule. In the graphical representation of the Ross Notation, the correspondent connection *exits* the rule symbol and *enters* the correspondent. Both the anchor connection and correspondent connection are dashed.

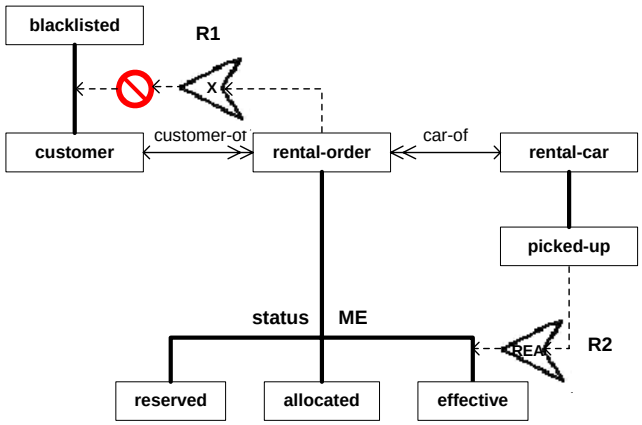


Figure 2-1. Representation of business rules by the Ross Notation.

The example represented in Figure 2-1 includes two integrity constraints: a rejector R1 of the type "Mandatory" with the meaning "Must have" and a projector R2 of the type "Enabled-with-reversal" which creates (i.e., enables, or switches "on") instances of the correspondent when an instance of the rule's anchor is created and "reverses" the state of the instances of the correspondent when the instance of the anchor is deleted. The rule R1 thus states: "A customer attached to a rental-order may not have the status blacklisted". The meaning of the rule R2 is: a rental-order has the status effective *if* and *until* a rental-car related to it has the status picked-up. The symbol **ME** in Figure 2-1 means that the statuses reserved, allocated, and effective are mutually exclusive.

The Ross Notation is a key deliverable of Proteus, the business rule methodology of Business Rule Solutions (BRS), LLC. In [Ross03] it is claimed that the Proteus methodology addresses the six different aspects of business modelling of the ISA framework, which were reviewed in section 1.5.2, in the following way:

- The ***motivation aspect*** is addressed by creating a *Policy Charter* which outlines the appropriate ends (e.g., business goals) and means (e.g., tactics) for solving the business problem.
- The ***function aspect*** is addressed by developing business process models that sequence the flow of tasks. According to [Ross03], the functional view is addressed by developing *scripts* which loosely integrate declarative business rules. A script is a procedure consisting of a series of requests for action to software components and/or humans with no embedded business rules [Ross03].
- The ***data aspect*** is addressed by developing the standard business vocabulary of the targeted business area, consisting of core concepts of the area. These definitions are organized into a *Concepts Catalog*, which is essentially a glossary of terms, and a *fact model* which complements it with relationships between the concepts.
- The ***people aspect*** is addressed by defining organizational roles and responsibilities, and the work relationships between them.
- The ***time aspect*** is addressed by examining the regimens needed to organize the aging of core concepts.
- The ***location aspect*** is addressed by building a *Business Connectivity Map* indicating business sites and their communication/transport links from the business perspective.

2.1.1.2. Evaluation

The Ross Notation is one of the most comprehensive representation formats for modelling business rules. Since the Ross Notation is largely a data- and database-oriented notation, it provides a very strong support for the informational view of business modelling. However, even though it is claimed in [Ross03] that the BRS business rules methodology also addresses all the other aspects listed above, the coverage of the functional view by the Ross Notation is weak. Also Hurlbut remarks in [Hurlbut98] that "the primary deficiency of the Ross Notation is its inability to model process aspects, due to its fundamental restriction of only considering persistent data as a basis for business rules". In [Ross03] it is claimed that loose scripts are sufficient for modelling business processes. However, there is neither syntax nor semantics included for such scripts in [Ross03]. This excludes a support for the behavioural view.

The motivational view is strongly supported due to using the Business Rule Motivation Model which was briefly described in section 1.4.1.2. However, the relationship between business goals and rules is not always clear in that model.

The organizational view is only present in scripts where a human actor can perform actions and also make requests for action to software components and to other actors.

Within the Ross Notation, *events* of the interactional view are understood as update events in a database and not events in a broader sense – *business events*. This also excludes proper treatment of interaction and communication between roles and actors.

In conclusion, we can claim that the Ross Notation and the Proteus methodology provide a very strong support for the informational view, a strong support for the motivational view, a weak support for the organizational and functional views, and virtually no support for the behavioural and interactional views.

The Ross Notation has also problems of a more fundamental nature. For example, it defines a 'condition' as a rule while it really is a *precondition* for a reaction rule and not a rule independently. The Ross Notation also calls a projection controller an 'integrity constraint' while it really seems to be a kind of derivation rule.

2.1.2. Eriksson-Penker Extensions to UML

According to the adaptation of the Unified Modelling Language (UML) for business modelling described in [Eriksson99] and [Eriksson00], called *Eriksson-Penker Business Extensions*, the primary concepts used when defining the business system are:

- ***Goals***. The purpose of the business and/or the outcome the business as a whole is trying to achieve. Goals can be broken down into sub-goals and allocated to individual parts of the business, such as processes or objects. Goals express the desired states of resources and are achieved by processes. *Goals can be expressed as one or more rules*.
- ***Resources***. The objects within the business, such as people, material, information, and products that are used or produced in the business. The resources are arranged in structures and have relationships with each other. Resources are manipulated (used, consumed, refined, or produced) through processes. Resources can be categorized into physical, abstract, and informational resources.
- ***Rules***. Statements that define or constrain some aspect of the business, and represent business knowledge. They govern how the business should be run (i.e., how the processes should execute) or how resources may be structured and related to each other.
- ***Processes***. The activities performed within the business in which the state of business resources changes. Processes describe how the work is done within the business. *Processes are governed by rules*.

These concepts are captured by four different views of a business used by the Eriksson-Penker Business Extensions. ***Business Vision View*** describes a goal structure for the company, and illustrates problems that must be solved in order to reach those goals. ***Business Process View*** illustrates the interaction between the processes and resources in order to achieve the goal of each process, as well as the interaction between different processes. ***Business Structural View*** describes the structures among the resources in the business, such as the organization of the business or the structure of the products created. ***Business Behavioural View*** models the individual behaviour of each important resource and process in the business model and how they interact with each other.

In the Eriksson-Penker Business Extensions, each view is expressed in one or more UML diagrams. The diagrams can be of different types, dependent upon the specific structure or situation in the business to be depicted. In particular, the *goal/problem model* of the Business Vision View is a UML object diagram that breaks down the major goals of the business into sub-goals, and indicates the problems that stand in the way of achieving those goals and actions required for achieving the goals. The *conceptual model* of the Business Vision View is a UML class diagram that defines important concepts and relationships in the business to create a common set of terminology. Figure 2-2, adapted from [Eriksson99], shows a *process diagram* based on a UML activity diagram with stereotypes from the Eriksson-Penker Business Extensions where the relationships between different processes are shown. This diagram also makes use of swim lanes, which are used to show the organization units involved in the process. Within the Business Structural View, organizational structure of the company is modelled by using *class* and *object diagrams* of UML. Finally, the Behavioural View makes use of UML state chart diagrams, sequence diagrams and collaboration diagrams for modelling each of the involved objects in more detail.

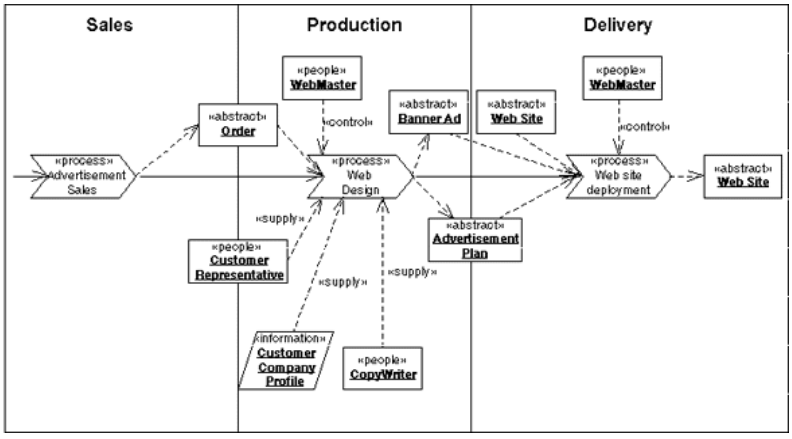


Figure 2-2. A process diagram based on an activity diagram with swimlanes.

2.1.2.1. Modelling of Business Rules

Eriksson and Penker, in [Eriksson00], follow [Martin98] in their classification of business rules, but add a third top-level category: 'existence rules' that define "under what circumstances something can exist".

Eriksson and Penker propose to express business rules using the Object Constraint Language (OCL) which is a part of the UML standard [OMG03a]. While this proposal seems natural for integrity constraints, its feasibility in the form it is expressed in [Eriksson00] is less clear for the other types of business rules.

For derivation rules, no general method how to express them in OCL is presented. In one example in [Eriksson00], a derivation rule for a derived Boolean attribute highRisk is expressed as a postcondition for the corresponding operation highRisk(). In another example, a derivation rule is expressed as an implication by means of the OCL implies connective. In still another example, a constraint is confused with a derivation rule.

Eriksson and Penker admit that 'stimulus/response' (i.e. reaction) rules cannot be defined in OCL because OCL cannot be used to define actions. They propose, instead, to define this type of rules in UML activity or state chart diagrams. Again, no general method is presented and only vague and somewhat confused indications are given (e.g., specifying the event condition of a 'stimulus/response' rule as guards in an activity diagram).

Finally, Eriksson and Penker argue that there are business rules which are best formalized using concepts from *fuzzy logic*. An example of such a rule would be one that defines a customer target group in terms of *middle-aged* persons with a *high salary*. According to [Eriksson00], both concepts are best captured by means of fuzzy-set-valued attributes.

2.1.2.2. Evaluation

In terms of the six views of agent-oriented modelling which we proposed in section 1.5.6, the Eriksson-Penker Business Extensions provide a strong modelling support for the informational view by object class diagrams and OCL expressions for integrity checking. However, Eriksson and Penker do not present a general method how to express derivation rules in a class diagram.

The Eriksson-Penker Business Extensions provide a strong support for the functional view and for the closely related to it behavioural view by employing (modifications of) activity diagrams and state chart diagrams. Eriksson and Penker claim that business processes can be modelled by UML activity diagrams as sequences of activities. But, as it has been noticed in [Eshuis02a]: "To ensure that every activity diagram can be translated into a state chart, UML 1.4 only allows activity diagrams in which each fork is eventually followed by a join and in which multiple layers of forks and joins are well nested". According to [Eshuis02a], such hierarchy constraints rule out certain forms of concurrency. UML 2.0, which is currently under development, will not adopt this constraint, but has other problems like separating data flow and control flow which may thus become inconsistent.

The Eriksson-Penker Business Extensions also support to the same extent the organizational view by representing organization models as class diagrams and organizations as object diagrams based on them. However, in the proposal by Eriksson and Penker, there is no specific treatment of agents. They are subsumed, together with "material, information, and products", under the concept of *resources*. This unfortunate subsumption of institutional and human agents under the traditional 'resource' metaphor prevents a proper treatment of many agent-related concepts such as commitments, deontic assignments, and communication/interaction.

Because agents/actors are not treated as first-class citizens, the representation of the interactional view is weak in the Eriksson-Penker Business Extensions. Another reason for the weakness of the interactional view is that it is questionable to model the invocations of activities in the spheres of responsibility of different "actor objects" as state transitions as it is done between 'swim lanes' in activity diagrams because the "actor objects" are in principle autonomous and independent of each other. The communication between different actors/agents should be modelled as a protocol instead [Sladek96]. To the limited extent, communication and interaction modelling is possible by using interaction diagrams of UML.

The motivational view is supported by the goal/problem model, which is essentially an object diagram, in the Eriksson-Penker Business Extensions. In spite of the claim in [Eriksson99] that in UML goals can be expressed as one or more business rules, it is not clear how the goal/problem model can be transformed into specific business rules.

2.1.3. Role Activity Diagrams

Role Activity Diagrams were proposed in [Ould95]. A ***Role Activity Diagram*** shows the roles, their component activities, and their interactions together with external events and the logic that determines what activities are carried out when [Ould95]. Role Activity Diagrams are based on *Petri Nets* which are described e.g. in [Aalst00].

2.1.3.1. Identification of Roles

In [Ould95], a ***role*** is defined as "an area of responsibility for some contribution to a process, carried out through a set of partially ordered activities which share a single role body or set of resources". According to [Ould95], roles can take many forms:

- A unique functional group, e.g. *accounts department*.
- A unique functional position or post, e.g. *managing director*.
- A rank or job title, e.g. *principal systems analyst*.
- A replicated functional group, e.g. *department*, *branch*.
- A replicated functional position or post, e.g. *head of department*, *branch manager*.
- A class of person, e.g. *customer*, *supplier*.
- An abstraction, e.g. *project managing*.

A role can have a number of instances at any one moment, and a ***role instance*** exists independently of the existence of an agent to play the role. The agent can change and at a given instant no one and nothing might be playing a given role instance [Ould95]. The key characteristics of the role forms presented above are summarized in Table 2-1 adopted from [Ould95].

Table 2-1. Characteristics of role forms.

Role form	*Number of instances*	*Permanent instances?*	*Can actor change?*
Unique functional group	1	+	+
Unique post	1	+	+
Job title	>1	-	-
Replicated functional group	>1	-	+
Replicated post	>1	+	+
Class of person	>1	-	-
Abstract role	>1	-	+

2.1.3.2. Notation

The notation for Role Activity Diagrams is presented in Figure 2-3. As Figure 2-3 shows, each role in the process is represented by the contents of a shaded block. Within each role, there are a number of activities indicated by black boxes, the annotation against each black box describing the activity succinctly using a verb. ***Activities*** are what agents do as individuals in their roles. An instance of an activity type is created when the organization's business process is in a particular condition that we call the ***activating*** or ***triggering condition*** for the activity. This is a *sufficient* condition for the activity instance to start. There may be other *necessary* conditions that are also true when an activity is started. These are collectively referred as the activity's ***precondition***. Similarly, each activity will have some ***post condition*** which describes the state of the world at the time the activity stops.

In the notation for Role Activity Diagrams depicted in Figure 2-3, *states* or conditions which a role can be in are represented by the vertical lines between activities within the role. The ***goal of a process*** is some point in the activity of a particular role where the state of the process is "goal achieved". In the notation for Role Activity Diagrams, the process goal is represented by putting a "magnifying glass" on the state line and annotating it, as can be seen in Figure 2-3.

An ***interaction*** between roles is shown as a white box in one role connected by a horizontal line to a white box in another role as is depicted in Figure 2-3. An interaction can involve any number of roles. Interactions in Role Activity Diagrams are modelled as *synchronous*: that is, all role instances must be ready for the interaction to take place before it can start, it starts at the same moment for each party, and it completes at the same moment for each party at which point they enter their respective new states.

In some situations it is useful to show which party to an interaction takes the lead or is responsible for making the interaction happen. Such a party is pointed out by shading in the interaction the white box for the driving role as is shown in Figure 2-3.

Alternative courses of action are represented with the notation shown in Figure 2-3 for *case refinement*. The two-way case refinement depicted in Figure 2-3 generalizes quite naturally to *N*-way case refinements.

Starting a number of separate threads of activity that can be carried out concurrently is represented by the structure shown in Figure 2-3 for *part refinement*.

An iteration is captured with the part refinement structure where the replication is indicated with an asterisk as can be seen in Figure 2-3.

An external event is shown by an arrow placed on the state line, as is shown in Figure 2-3. An external event may also represent calendar or clock time or the passage of time.

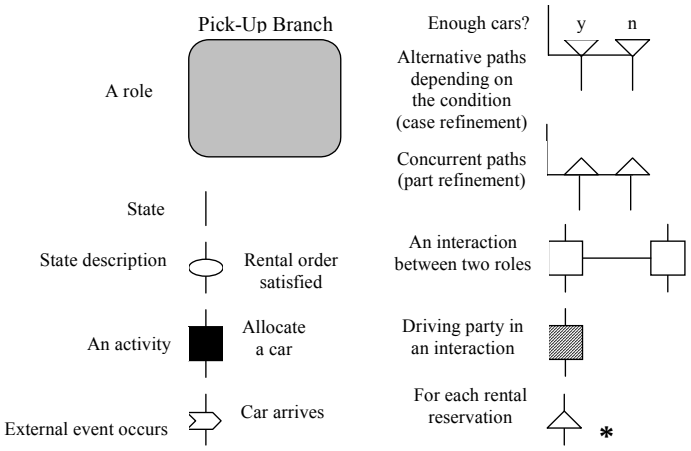


Figure 2-3. The notation for Role Activity Diagrams.

2.1.3.3. Evaluation

In terms of the views of agent-oriented modelling which were presented in section 1.5.6, Role Activity Diagrams provide a very strong support for the functional view and a strong support for the organizational view. At the same time, Role Activity Diagrams do not support to any extent the informational view. This conclusion is shared in [Curtis92], where it is stated: "Role Activity Diagrams are strong in representing roles, dependencies, and process elements but its representation of artifacts is weak". The lack of support for the informational view causes the weakness of the motivational view because there is no vocabulary in which process goals could be precisely represented. The mentioned deficiency is also reflected on the behavioural view because behavioural constructs like part refinement can not be connected to data elements and data flow. For this reason, the Role Activity Diagrams do not enable precise behaviour modelling.

According to [Curtis92], when a process has been represented using Role Activity Diagrams, a new notion of teams emerges, built on dependencies among roles. Where interactions among roles are frequent, a clustering of roles forms a *de facto* team. The support for the organizational view by Role Activity Diagrams, however, lacks the means for representing static organization structures which include relationships between organizational units like aggregation.

Another shortcoming of Role Activity Diagrams related to the interactional view is their inability to model asynchronous communication. This prevents the use of Role Activity Diagrams for modelling numerous real-life communication situations, including the situations where software agents or other automated systems with message buffering capability are involved.

2.1.4. *i** and Tropos

The ***i**** framework was proposed in [Yu95a] and [Yu95b]. The ***i**** (which stands for "distributed intentionality") framework provides understanding of the motivation of social actors that depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. The framework consists of a ***Strategic Dependency*** (***SD***) model and a ***Strategic Rationale*** (***SR***) model. The SD model provides an intentional description of a (business) process in terms of a network of dependency relationships among actors. The SR model provides an intentional description of a (business) process in terms of process elements and the rationales behind them [Yu95a].

2.1.4.1. Analysis of Dependencies

Four types of dependencies are distinguished among actors, based on the type of dependum. In a ***goal dependency***, a depender depends on the dependee to bring about a certain state in the world. The dependee is given the freedom to choose how to do it. Under goal dependency, the dependee is free to, and is expected to, make whatever decisions that are necessary to achieve the goal. In a ***task dependency***, a depender depends on the dependee to carry out an activity. A task dependency specifies *how* the task is to be performed, but not *why*. The depender's goals are not given to the dependee. In a ***resource dependency***, the depender depends on the dependee for the availability of an entity (physical or informational). By establishing this dependency, the depender gains the ability to use this entity as a resource. Under resource dependency, it is assumed that there are no open decisions to be addressed by the dependee. In a ***softgoal dependency***, a depender depends on the dependee to perform some task that meets a *softgoal*. The meaning of the softgoal is not clear-cut. It is specified in terms of the methods that are chosen in the course of pursuing the goal.

The intentional dependencies of the domain of car rental are represented in Figure 2-4.

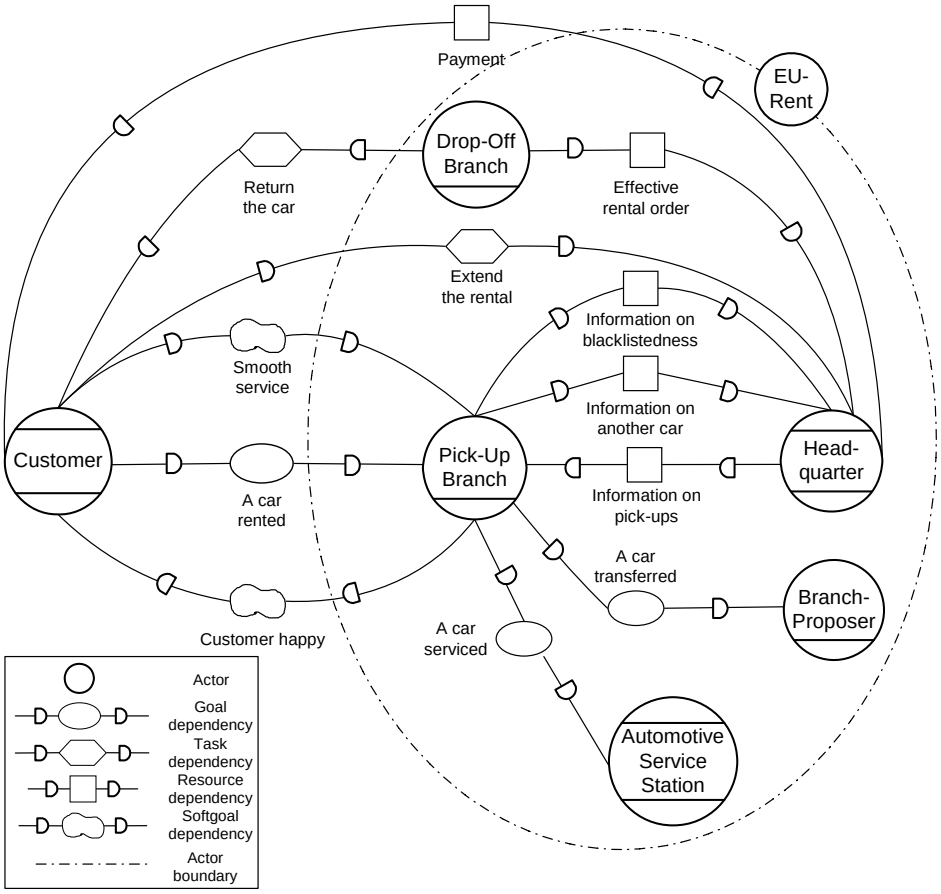


Figure 2-4. The Strategic Dependency Model for the domain of car rental.

2.1.4.2. Means-Ends Analysis

The SR model provides an intentional description of a (business) process in terms of process elements and the rationales behind them. While the SD model represents only the external relationships between actors, the SR model describes the intentional relationships that are internal to actors, such as means-ends relationships.

There are two main classes of links: means-ends links and task decomposition links. A ***means-ends link*** indicates a relationship between an end – which can be a goal to be achieved, a task to be accomplished, a resource to be used, or a softgoal to be satisfied – and a means for attaining it. The means is usually expressed in the form of a task, since the notion of task embodies *how* to do something.

A task node is linked to its component nodes by ***task decomposition links***. There are four types of task decomposition links – *sub-goal*, *subtask*, *resourceFor*, and *softgoalFor* – corresponding to the four types of nodes. These links can also connect up with dependency links in SD model(s), when the reasoning goes beyond an actor's boundary.

For example, the SR model for the Pick-Up Branch shows that the Pick-Up Branch is able to achieve the goal "*A car rented*" that the Customer depends on by running the task "*Rent a car*", whose goal is to provide the *Customer* with a car. This task consists of four components: the subtasks "*Receive rental order*" and "*Manage rental reservation*", the sub-goal "*A car allocated*" (a car is allocated to the rental order 12 hours before the pick-up time), and the subtask "*Deliver the car*". The model also includes the alternative means how to achieve the sub-goal "*A car allocated*" depending on the availability of cars.

The ***i**** notation is accompanied by the ***Tropos methodology*** [Mylopoulos01] which consists of the following steps: *early requirements acquisition with **i****, resulting in SD and SR models of the kind described above, *definition of late requirements in **i****, where the system-to-be is described within its operational environment, *architectural design using **i****, where the system's global architecture is defined in terms of subsystems, and *detailed design* where the behaviour of each architectural component is defined in further detail.

In the example of the car rental, the phase of defining late requirements in ***i**** produces revised SD and SR models which include a computerized agent-based system for the car rental. The system is represented as one or more actors participating in a SD model, along with other actors from the system's operational environment. At this stage, the system is also decomposed into several sub-actors using the same kind of means-ends analysis as in the early requirements acquisition described above. At the stage of architectural design, a proper architectural style of the agent-based system is selected from among alternative ones like e.g. *flat structure*, *pyramid*, *joint venture*, and *structure-in-5*. The analysis involves refining the desired qualities of the system, represented as softgoals, to sub-goals that are more specific and more precise and then evaluating alternative architectural styles against them. Finally, at the stage of detailed design, the SD and SR models are transformed into Agent Class Diagrams, Sequence Diagrams, Collaboration Diagrams, and Plan Diagrams by using AUML [Odell00].

2.1.4.3. Evaluation

The ***i**** is targeted at early requirements engineering. It emphasizes the motivational view of agent-oriented modelling and provides a strong support for the organizational and functional views and some support for the informational view. Within the organizational view, other kinds of relationships between organizational units besides dependencies, like aggregation, are not analyzed. The functional and informational views are supported through representing tasks and informational resources, respectively. The ***i**** also supports to some extent the interactional view because representing intentional dependencies can be viewed as an early stage of interaction modelling.

The SR diagram notation of ***i**** represents a sequence of tasks (activities) to be performed by an individual agent, but it is imprecise with regard to representing models of choices between alternative courses of tasks, models of concurrent threads of tasks, and iteration models of tasks. For example, it is not possible to specify whether the subtask in a task decomposition link has to be performed once or several times. In other words, ***i**** does not support the behavioural view of agent-oriented modelling. Some authors have suggested using SR diagram annotations to cope with the deficiencies mentioned. E.g., in [Wang01], two types of annotations are defined: composition annotations and link annotations corresponding to the operators of the ConGolog specification language [Lesperance99] and having the same meaning.

The Tropos methodology focuses on the organizational and motivational views and introduces a business vocabulary (ontology) of the informational view only at the stage of detailed design. In our opinion, in order not to run into inconsistencies between models of different views, the modelling of objects of the problem domain and relationships between them should be started at least at the beginning of the design phase.

We have also discovered that in modelling real-life situations, it can be really confusing to distinguish between goal and task dependencies. For example, in case of a car rental company, if a customer's interface to the company is well-defined, we can model the dependency between the customer and the company as a task dependency. Otherwise, we should model it as a goal dependency.

2.1.5. CIMOSA

CIMOSA is an open system architecture which has been developed for integration in manufacturing but which is widely applicable to integration of any type of enterprises [AMICE93]. CIMOSA is the result of a joint research and development effort by more than 30 European companies over the period 1985–1994 funded in-half by EU.

One of the major achievements of the AMICE Consortium was the development of the CIMOSA language for enterprise modelling. This language complies with the enterprise modelling principles, generally represented by a cubic structure. According to [Berio99], the foundations of all generic architectures, represented by the CIMOSA Cube, show that any approach for enterprise modelling must at least deal with three fundamental types of flows within or across enterprises (***material flows***, ***information flows***, ***decision/control flows***), four modelling views (***function view***, ***information view***, ***resource view***, ***organization view***), and three modelling levels (***requirements definition***, ***design specification***, ***implementation description***). The CIMOSA modelling language [Vernadat98] provides constructs for function, information, resource and organization aspects in a unified formalism. The language uses an event-driven process-based approach described in [Vernadat98]. According to [Berio99], in the CIMOSA language, an enterprise is composed into a set of ***domains*** which are functional areas of the enterprise. A ***domain process*** is a complete chain of activities flowing through the enterprise irrespective of organizational boundaries. It is triggered by one or more ***events*** and terminates when it produces a definite desired end-result. A domain process is made up of sub-processes, called ***business processes***, which, in turn, consist of ***enterprise activities***. Domain and business processes and enterprise activities are subject to ***objectives*** and/or ***constraints***. Enterprise activities require ***resources*** and time to transform states of ***enterprise objects*** into different states. These states are called ***object views***.

CIMOSA differentiates active resources (called ***functional entities***), which have capabilities and can provide services, as opposed to inactive resources (called ***components***) which are utilized by functional entities. Three types of functional entities are distinguished in CIMOSA: humans, machines, and applications. A functional entity provides a ***capability set*** which is required by one or more enterprise activities.

The organizational view of CIMOSA enables to model the ***organization units*** of the enterprise and their relationships, as well as distribution of responsibilities and authorities between them.

According to [Reyneri99], in CIMOSA Petri nets are used for modelling resource behaviour, while in [Berio99] it is suggested to use state-transition diagrams and especially statecharts for the same purpose, because "the concept of states is a fundamental feature of resources" [Berio99]. Moreover, according to [Berio99] "statecharts can be used to model the behaviour of a single resource or the interaction between several interacting resources". For less structured resource interactions, such as for the modelling of human agent communication in team-working, it is proposed in [Berio99] to use the Speech–Act Perspective as suggested by Medina-Mora et al. [Medina-Mora92].

2.1.5.1. Modelling of Business Rules and Processes

Functional modelling in CIMOSA addresses both enterprise *functionality* described in terms of enterprise activities and hierarchy of functions and enterprise *behaviour* described in terms of business processes.

According to [Berio99], business processes are usually defined in the form of a workflow or partially ordered set of process steps, which *in fine* is equivalent to a network of enterprise activities. The workflow represents the control flow of the process and is defined by means of a set of connectors which can be junction boxes, behavioural rules, or temporal logic operators. These connectors are used

to define control structures in the workflow (such as process start, sequence, branching, spawning, rendezvous, loop, and process end).

A restricted form of reaction rules, called 'procedural rules' (and more recently 'behavioural rules'), is used in CIMOSA to specify control structures for business processes. These rules have the form

```
WHEN event DO action
```

where the event expression typically refers to the ending status of some activity, such as in the following rule

```
WHEN ES(ea1) = ok DO ea2.
```

specifying that the enterprise activity `ea2` is started when the ending status of the enterprise activity `ea1` is '`ok`'.

The reaction rules of CIMOSA also enable to model nondeterministic enterprise behaviour where there are choices in the control flow of a business process left open to an external agent. The CIMOSA constructs for modelling nondeterministic enterprise behaviour correspond to the general behavioural construct "Deferred choice", which is described in [Patterns03].

According to [Berio99], CIMOSA provides four ways for synchronization of business processes:

- *synchronization by events* (one activity in a process P1 generates an event Ev1 which triggers another process P2, either in the same domain or in another domain);
- *synchronization by object availability*: the output of an activity of process P1 can be the input of an activity of process P2;
- *synchronization by resource availability* (resources are allocated to processes on the basis of schedules or priority rules);
- *synchronization by message passing*.

In CIMOSA, both synchronous or asynchronous communications between activities are allowed. They are made by message passing using the following pre-defined functional operations where *m* is a message and *a* an activity:

- *send* (*a*, *m*): to send a message *m* to activity *a*;
- *receive* (*a*, *m*): to receive a message *m* from activity *a*;
- *acknowledge* (*a*): to let activity *a* know that the message was received;
- *broadcast* (*m*): to send a message *m* to anyone who wants to read it.

2.1.5.2. Evaluation

CIMOSA provides a strongest support for the functional and behavioural views of agent-oriented modelling. It also supports strongly the informational view, which, however, does not include constructs corresponding to derivation rules, and provides some support for the motivational view. The support for the organizational and interactional views is weakened by the incorrect and unsystematic treatment of actors/agents. Like in Eriksson-Penker extensions to UML (v. section 2.1.2), in CIMOSA agents are understood as "active resources, able to perform a number of atomic actions, called functional operations". This definition is made even fuzzier in [Reyneri99], where actors are defined as "all elementary elements [of a domain model]: enterprise activities, organization units, humans, single machines, information elements…". It is hard to imagine something in common between the things mentioned.

The applicability of Petri Nets for modelling resource behaviour mentioned in [Reyneri99] is questionable in case of open systems, because, according to [Eshuis02a] "all changes in Petri nets occur because of the firing of some transitions in the net that represent activity of some part of the system itself, rather than some activity in the system's environment". In other words, Petri nets are suitable for modelling active systems, rather than reactive ones.

Interaction modelling by statecharts proposed in [Berio99] suffers from the same problem as activity diagrams of UML 1.4 which is formulated in [Sladek96] with regard to the case of inter-organizational business processes studied in the paper: "It was incorrect to model the task activation across the organizational boundaries as a state transition because it meant that the requesting organization was loosing the control and thus its autonomy also. To alleviate the problem, the inter-organizational communication should be modelled as a protocol". This applies to interactions between different units, e.g. departments, of an organization, as well as to interactions between different organizations.

In CIMOSA, events accepted by internal behaviours of a business process are represented as occurrences of different ending statuses of the process [Berio99]. It is confessed in [Berio99]: "Obviously, this constrains the interface between the process instance and its execution environment". Modelling of interactions by speech acts is mentioned in [Berio99], but it is not made clear how this kind of interaction modelling can be integrated into statechart-based modelling. The representation of the interactional view is thus weak in CIMOSA.

2.2. METHODOLOGIES FOR BUSINESS MODELLING

2.2.1. Business Rule-Oriented Conceptual Modelling

Business **R**ule-**O**riented **CO**nceptual **Modelling** (**BROCOM**) was proposed in [Herbst95] and [Herbst97]. The BROCOM approach emphasizes the use of business rules for the specification of all dynamic properties relevant to the universe of discourse, i.e., of processes and integrity constraints. The methodology is based on the metamodel that consists of the following sub models: 'Business Rule', 'Data Model Components', 'Processor', 'Origin', 'Organizational Unit', and 'Process'.

The sub model 'Business Rule' consists of the four meta entity types: *Business rule*, *Event*, *Condition*, and *Action*. Every business rule has exactly one event, at most one condition, and one or two actions (*THEN* / *ELSE*). Events and conditions may be composite and therefore have recursive M:N relationship types. Actions of business rules may raise events. In the metamodel, this is represented by the relationship ***is_raised_by*** between the meta entity types *Action* and *Event*.

To illustrate the scope and different types of business rules, some examples which may be relevant in an order processing system are introduced in [Herbst95]. In the following business rules, after the customer has specified an order, the order is only accepted if the total amount of the order does not exceed the actual credit limit of the customer. The acceptance of the order results in triggering the tasks of assembling and delivering the order:

[BR2] *ON order specified*
IF (credit-limit of customer > order-total)
THEN registrate order, ⇒ EVENT 'order registrated'
SET order-state := 'accepted'
SET credit-limit := credit-limit - order-total
ELSE reject order, ⇒ EVENT 'order rejected'

[BR4] *ON order registered*
THEN assemble order, ⇒ EVENT 'order assembled'

[BR6] *ON order assembled*
THEN deliver order, ⇒ EVENT 'order delivered'

The sub model 'Data Model Components' encompasses the meta entity types for a conceptual data model. In accordance with the Entity Relationship Model, the meta entity types ***Entity type***, ***Relationship type***, and ***Attribute*** are incorporated into this sub model. The allowed semantics of ***references*** relationship between components of business rules and data model components is put together in Table 2-2, adopted from [Herbst97]. For example, the impact of the rule *[BR2]* on data model components is insertion of a new order and modification of the order state. And the other way round, the data model component (entity type) 'Order' is referenced by the rule *[BR2]*.

Table 2-2. Relationship between business rules and modelling constructs.

Relationship from	Retrieval	Modification
Event ⇒ Data model component	No	No
Condition ⇒ Data model component	Yes	No
Action ⇒ Data model component	Yes	Yes

The sub model 'Processor' links rule components to specific processors who/which execute them, i.e., who/which detect the occurrence of events, evaluate conditions, and perform actions. A 'Processor' can be a ***human actor***, ***machine***, or ***software program***. A 'Processor' raises 0 to N events, evaluates 1 to N conditions, and performs 0 to N actions.

Business rules may *originate* outside or inside an organization which is addressed by the sub model 'Origin'. Externally originating rules can be further divided into ***natural facts*** which are eternally

fixed and (e.g., legal) ***norms*** which are specified by the society and may change. Internal origins can be either *primary* or *secondary*; an origin is primary if its content is originally described in a source document, whereas a secondary origin has previously been derived from another source.

Within the sub model 'Organizational Unit', the assignment of business rule components to the ***organizational units***, which are responsible for processing the components, leads to *intra* and *inter* unit rules. This classification may help to support the administration of business rules in an organization. Organizational units ***own*** origins and ***encompass*** processors of business rules.

Within the sub model 'Process', actions of business rules can be related to events resulting in ECA-chains describing the dynamic of ***processes*** like the example depicted in Figure 2-5, adopted from [Herbst95], which also includes the example rules *[BR2]*, *[BR4]*, and *[BR6]*, presented above. Processes can thus be specified by means of business rules. In the context of the metamodel only the behaviour of a business process is considered. Additional properties like process goals, values, and process owners are not further discussed.

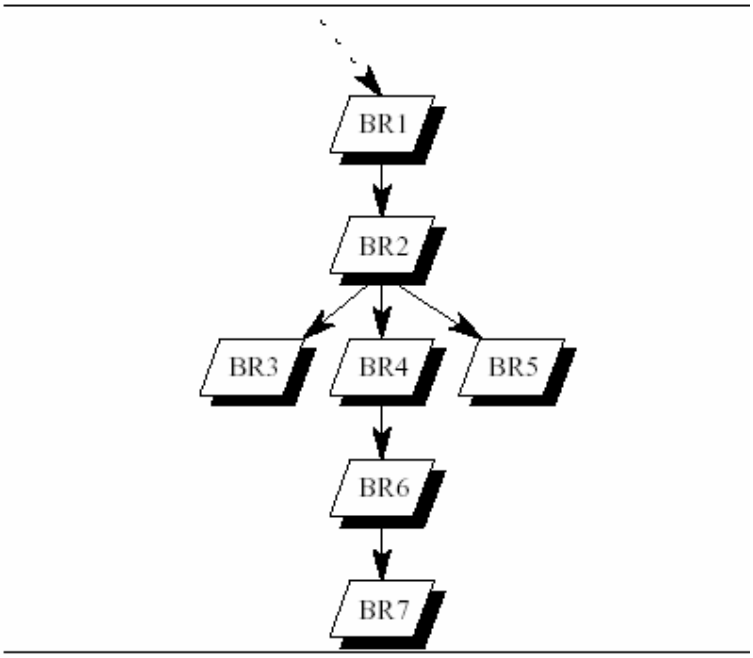


Figure 2-5. Order processing described by business rules

In [Herbst97], the following five modelling steps are proposed:

1. Specification of the process structure.
2. Specification of the processes by using business rules.
3. Specification of the conceptual data model.
4. Specification of integrity constraints by using business rules.
5. Validation.

Steps one and two concern *process specific business rules* and steps three and four *process independent business rules*.

2.2.1.1. Evaluation

The BROCOM approach as a database-oriented methodology provides a very strong support for the informational view of agent-oriented modelling. It also integrates the informational view with the functional and organizational views. The functional and organizational view are thus supported strongly but the behavioural view only weakly, because BROCOM does not include any explicit behavioural constructs, even though some of them can be simulated. There is no support for the motivational view for the reason that, even though process goals are mentioned in [Herbst97], in reality they are not represented in BROCOM.

The BROCOM's sub model 'Processor' includes both human and automated actors. However, the BROCOM approach does not include the notion of communication/interaction, even though it acknowledges that actions performed by actors raise events. For example, raising the event 'order assembled' by the rule *[BR4]*, which occurs in the storage department, and the reaction to this event

by the rule *[BR6]* performed in the sales department could be more naturally modelled as sending a message from the storage department to the sales department, especially if they lie in different geographical locations. Likewise, raising the event 'order delivered' within the business rule *[BR6]* could be modelled as an interaction (providing the commodity requested) between the sales department and the customer.

2.2.2. Enterprise Knowledge Development (EKD)

The Enterprise Knowledge Development (EKD) methodology is comprehensively presented in [Bubenko01], according to which the purpose of applying EKD is to provide a clear, unambiguous picture of how the enterprise functions currently, what are the requirements and the reasons for change, what alternatives could be devised to meet these requirements, and what are the criteria and arguments for evaluating these alternatives. Basic contents of the EKD framework include: a *set of description techniques*, *explanation of stakeholder participation*, and a *set of guidelines for working*. EKD application process is supported by a *set of software tools*.

The deliverables of the EKD process are a number of *conceptual models* that examine an enterprise and its requirements from a number of interrelated perspectives. These models are based on the refinement of the enterprise model that was discussed in Section 1.5.3. The refined enterprise model is depicted in Figure 2-6 which is adapted from [Bubenko01]. As the figure shows, it contains a number of sub models which are connected to each other by *inter-model relationships*. Each of the models represents some aspect of the enterprise.

The ***Goals Model*** is used for describing the goals of the enterprise along with the issues associated with achieving these goals. Component types of the Goals Model are *goal*, *problem*, *cause*, *constraint*, and *opportunity*. The link types between the components of the Goals Model are *supports*, *hinders*, and *conflicts*.

The ***Concepts Model*** is used to define the concepts of the problem domain and the attributes that characterize them. Concepts can be related to each other by means of *binary relationships*, *generalization/specialization relationships*, and *aggregation relationships*.

The ***Business Rules Model*** has the central position among other types of conceptual models in Figure 2-6. It is used to define and maintain explicitly formulated business rules, consistent with the Goals Model. ***Business rules*** are defined in EKD as "the rules that control the enterprise in a way that they define and constrain which actions may be taken in the various situations that may arise" [Bubenko01]. According to [Bubenko01], business rules may be in the form of precise statements that describe the way that the business has chosen to achieve its goals and to implement its policies, or the various externally imposed rules on the business, such as regulations and laws.

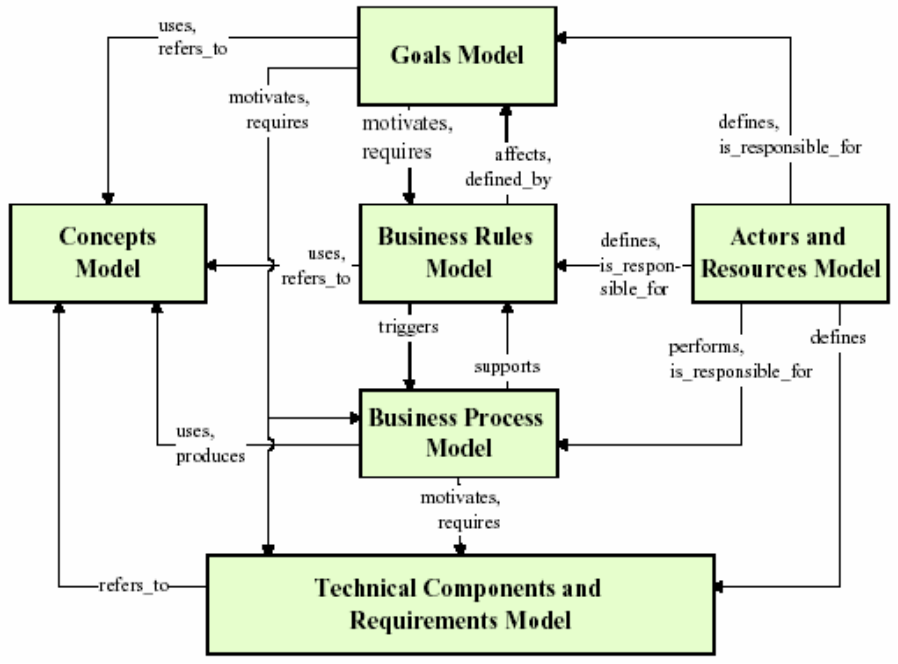


Figure 2-6. The submodels comprising the refined Enterprise Model.

In [Bubenko01], business rules are categorized into ***derivation rules***, ***event-action rules***, and ***constraint rules*** corresponding to derivation rules, reaction rules, and integrity constraints which were defined by us in section 1.4.1.1. Event-action rules in the Business Rules Model should be expressed in the following way: **When** {event} **If** {preconditions on entities} **then** {processes}. In Figure 2-7, adapted from [Bubenko01], a business rule of the library case study corresponding to the above pattern is visualized jointly with the concepts of the Concepts Model it refers to and processes of the Business Processes Model it motivates or is supported by. In the same way, business rules are also related to goals of the Goals Model.

The ***Business Processes Model*** is designed for analyzing the processes and flows of information and material in the enterprise. Processes can be decomposed into subprocesses. The Business Process Model also enables to model control flows using AND-join, OR-join, and OR-split constructs.

The ***Actors and Resources Model*** of EKD distinguishes between ***actors*** of the following kinds: ***individuals***, ***organizational units***, and ***roles***. Within the same model, ***non-human resources*** are understood as types of machines, systems of different kinds, equipment, etc. Binary relationships, generalization/specialization relationships, and aggregation relationships between actors and/or non-human resources are also a part of the Actors and Resources Model.

In [Kavakli98], *actor-role diagrams* were proposed for analyzing relations between business goals and business processes. An actor-role diagram presents a high-level view of the association between actors and their different roles, like the actor Customer Service Section and its role Service Administrative Handling. Actor-role diagrams enable to represent textually goals assigned to roles, like "Deal with contractual and financial matters" of the role Administrative Handling, and graphically different kinds of *dependencies* between roles, like authorization and co-ordination dependencies.

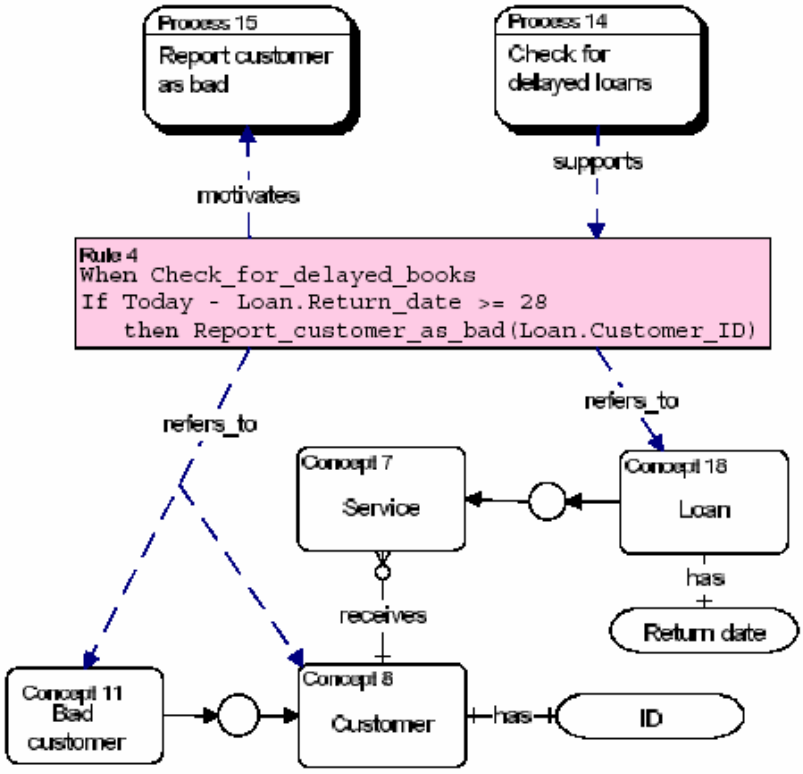


Figure 2-7. Rules refer to concepts in the Concepts Model and are supported by processes in Business Processes Model.

2.2.2.1. Evaluation

The EKD methodology provides a strong support for the informational, functional, organizational, and motivational views of agent-oriented modelling. As in ***i****, which was discussed in section 2.1.4, the interactional view is supported only indirectly and weakly through various dependency relationships between the actor types involved. Since the number of behavioural constructs in EKD is very limited, the support provided for the behavioural view is also weak.

Visualization of conceptual models in EKD is quite simplistic (by boxes). With the exception of role-activity diagrams, the EKD approach does not include any notation for more specific graphical representation of the components of one or another sub model and especially of the links between the components, as well as of the links between different sub models.

The EKD approach does not address the conceptual models and dependencies between them at the design level. Therefore it can only be used only at the requirements' analysis stage, while other, more precise techniques, should be used at the design stage.

2.2.3. Gaia

In Gaia [Wooldridge00], agents are understood as "coarse-grained computational systems, each making use of significant computational resources". Gaia is thus intended to be a software engineering technique, rather than a business modelling methodology. However, developing business process management systems as one of its application areas indicates that Gaia can (should) be used to some extent for business modelling, as well.

According to [Wooldridge00], Gaia includes a number of analysis and design models which are based on the notion of *role*. A ***role*** in Gaia is defined by four attributes: *responsibilities*, *permissions*, *activities*, and *protocols*. ***Responsibilities*** define the functionality of a role. They are divided into two types: *liveness properties* and *safety properties*. These notions have been borrowed from the theory of reactive systems presented in [Manna92]. ***Liveness properties*** intuitively state that "something good happens". They describe those states of affairs that an agent must bring about, given certain environmental conditions. In contrast, ***safety properties*** are *invariants*. Intuitively, a safety property states that "nothing bad happens", i.e. that an acceptable state of affairs is maintained across all states of execution.

In order to realize its responsibilities, a role has a set of *permissions* which are the "rights" associated with a role. The ***permissions*** of a role thus identify the (information) resources that are available to that role in order to realize its responsibilities. For example, a role might have associated with it the ability to read a particular item of information, or to modify another piece of information. A role can also have the ability to *generate* information. The ***activities*** of a role are computations associated with the role that may be carried out by the agent without interacting with other agents. Activities are thus "private" actions, in the sense of [Shoham93]. Finally, a role is also identified with a number of *protocols*, which define the way that it can interact with other roles. For example, a "seller" role might have the protocols "Dutch auction" and "English auction" associated with it.

According to [Wooldridge00], the objective of the analysis stage is to develop an understanding of the system and its structure (without reference to any implementation detail). This understanding is captured in the system's *organization*. The organization model in Gaia is comprised of the *roles model* and the *interaction model*.

The ***roles model*** in Gaia is comprised of a set of ***role schemata***, one for each role in the system. An example of a role schema is provided in Figure 2-8 which is adapted from [Wooldridge00]. This schema models the role **CoffeeFiller** whose purpose is to ensure that a coffee pot is kept full of coffee for a group of workers.

As can be seen in Figure 2-8, Gaia makes use of a formal notation for expressing permissions. For example, in Figure 2-8 two permissions are defined: the first says that the agent carrying out the role **CoffeeFiller** has permission to read the *coffeeMaker* parameter (that indicates which coffee machine the role is intended to keep filled). The agent has also permission to access the value *coffeeStatus* (that indicates whether the machine is full or empty) and to both read and modify the value *coffeeStock*.

In Gaia, liveness properties of a role are specified via a liveness expression, which defines the "life-cycle" of the role. The liveness expressions in Gaia are essentially regular expressions with an additional operator, "ω" for *infinite repetition*. For example, the liveness expression, which specifies in Figure 2-8 the responsibilities of the **CoffeeFiller** role, says that **CoffeeFiller** consists of executing the protocol Fill, followed by the protocol InformWorkers, followed by the activity CheckStock and the protocol AwaitEmpty. The sequential execution of these protocols and activities is then repeated infinitely often.

Safety properties in Gaia are specified by means of a list of predicates. These predicates are typically expressed over the variables listed in a role's permissions attributes. For example, the role schema depicted in Figure 2-8 includes the safety property *coffeeStock* > 0 which must be true across all states of the system's execution.

Role Schema: **CoffeeFiller**		
Description:		
This role involves ensuring that the coffee pot is kept filled, and informing the workers when fresh coffee has been brewed.		
Protocols and Activities:		
Fill, InformWorkers, CheckStock, AwaitEmpty		
Permissions:		
	reads	supplied *coffeeMaker* // *name of coffee maker*
		coffeeStatus // *full or empty*
	changes	*coffeeStock* // *stock level of coffee*
Responsibilities		
Liveness:		
CoffeeFiller = (Fill.InformWorkers.CheckStock. AwaitEmpty)$^{\omega}$		
Safety:		
coffeeStock > 0		

Figure 2-8. Schema for the role **CoffeeFiller**.

The ***interaction model*** in Gaia represents dependencies and relationships between the various roles in a multi-agent organization. This model consists of a set of *protocol definitions*, one for each type of inter-role interaction. According to [Wooldridge00], here a protocol is understood as a pattern of interaction that is abstracted away from any particular sequence of execution steps, like the precise ordering of particular message exchanges. For example, the Fill protocol, which forms a part of the **CoffeeFiller** role modelled in Figure 2-8, specifies that it involves **CoffeeFiller** putting coffee in the machine named *coffeeMaker*, and results in CoffeeMachine being informed about the value of *coffeeStock*.

Once all the roles and their interactions are captured, the *design process* can start. The aim of design in Gaia is to transform the analysis models into a sufficiently low level of abstraction that traditional modelling techniques (including object-oriented techniques) may be applied in order to implement agents. The Gaia design process involves generating three models: the *agent model*, the *services model*, and the *acquaintance model*.

The purpose of the Gaia ***agent model*** is to identify the various agent types that will make up the system under development, and the agent instances that will realize these agent types at run-time. The agent model is defined using a simple *agent type tree* mapping roles into agent types, in which leaf nodes correspond to roles (as defined in the roles model), and other nodes correspond to agent types. The agent model also specifies the number of instances of each agent type that will appear in a system. For example, the agent model of the coffee brewing example expresses that the agent role **CoffeeFiller** is mapped to the agent type FillerAgent, of which there are zero or more instances.

The ***services model*** describes the *services* associated with an agent role which are essentially the main *functions* that are required to realize the agent role. For each service that may be performed by an agent, its *inputs*, *outputs*, *preconditions*, and *postconditions* are identified. For example, in the coffee brewing example, there are four activities and protocols associated with the **CoffeeFiller** role: Fill, InformWorkers, CheckStock, and AwaitEmpty. In general, there will be at least one service associated with each protocol. In the case of CheckStock, for example, the service (which may have the same name), will take as input the stock level and some threshold value, and will simply compare the two. The pre- and postconditions will both state that the coffee stock level is greater than 0. This condition is one of the safety properties of the role **CoffeeFiller**.

Finally, ***acquaintance models*** simply define the communication links that exist between agent types. They do not define what messages are sent or when messages are sent – they simply indicate that communication pathways exist. Agent acquaintance models are directed graphs, and so an arc $a \rightarrow b$ indicates that a will send messages to b, but not necessarily that b will send messages to a. For example, the acquaintance model defined for the coffee brewing example states that the FillerAgent exchanges messages with the Machine Agent which, in turn, exchanges messages with the WorkerAgent.

2.2.3.1. Evaluation

In Gaia, the main emphasis of domain modelling is on the organizational, interactional, and functional views of agent-oriented modelling which are addressed by role and interaction models. However, in role modelling, Gaia does not provide any means for expressing other kinds of relationships between the roles apart from interactions, like generalization, aggregation, and control (subordination), which makes the overall support for the organizational view weak. This is not a surprise given that Gaia is aimed at developing agent-based software systems and not for modelling of business domains. The support for the interactional view is also not the best possible one because Gaia does not enable to model the types and contents of agent messages in agent protocols like "Dutch auction" and "English auction". Neither supports Gaia the modelling of the order in which agent messages are exchanged.

The purpose of Gaia (developing software systems) does not explain the lack of explicit modelling of agents' private and common knowledge in it, because a system of software agents also needs a common framework of knowledge – *ontology*. At present, the knowledge maintained by agents can be modelled by using role variables like *coffeeStatus* and *coffeeStock* and safety properties which is not sufficient for covering the informational view.

Gaia provides a strong support for the functional view through the permissions' and responsibilities' modelling at the stage of analysis and the modelling of inputs, outputs, and pre- and postconditions at the design stage. While the liveness expressions enable to specify the order in which protocols and activities are executed and their repetitions, they do not provide any means for expressing more complicated behavioural constructs where the number of repetitions depends on the value of one or more data items, like "Exclusive choice" and loops. This again reflects the insufficient support for the informational view of agent-oriented modelling by Gaia. Since, as we will see in section 3.6.2, postconditions are subsumed by goals, Gaia also provides some support for the motivational view of agent-oriented modelling.

We can conclude by saying that many deficiencies of Gaia that were discussed stem from the purpose of Gaia which is developing agent-based software systems rather than developing information systems based on business modelling.

2.3. COMPARISON OF THE BUSINESS MODELLING TECHNIQUES

In sections 2.1 and 2.2, five modelling languages and notations for business modelling and three business modelling methodologies were reviewed and evaluated with respect to the six views of agent-oriented modelling proposed by us in section 1.5.6. The results of the evaluation have been summarized in Table 2-3 where the following legend is used:

- - : no support;
- + : a weak (some) support;
- ++ : a strong support;
- +++ : a very strong support.

Table 2-3. Comparative evaluation of the business modelling techniques.

	Informational	Organizational	Interactional	Functional	Motivational	Behavioural
Ross Notation	+++	+	-	+	++	-
Eriksson-Penker Business Extensions	++	++	+	++	++	++
Role Activity Diagrams	-	++	++	+++	+	++
i*	+	++	+	++	+++	-
CIMOSA	++	++	+	+++	+	+++
BROCOM	+++	++	-	++	-	+
EKD	++	++	+	++	++	+
GAIA	+	+	++	+++	+	+

As Table 2-3 reveals, no one of the business modelling techniques studied in sections 2.1 and 2.2 provides a sufficient support for all views of agent-oriented modelling. Especially can be noticed the weakness of the interactional view in the modelling techniques and methodologies analyzed. This can be explained by the fact that up to the latest time business modelling has been aimed at creating monolithic information systems consisting of a thick server and many thin clients as opposed to truly distributed information systems consisting of subsystems which interact and communicate in a *peer-to-peer* manner. The results of the comparison have thus convinced us of the need to devise a business modelling notation and the relevant methodology that would be aimed at creating highly distributed agent-oriented and cooperative information systems. The technique, which we have named the ***Business Agents' Approach***, will be presented in Chapter 3 of this thesis.

2.4. OTHER RELATED WORK

Lately, a variety of XML-based techniques and notations for creating executable business process specifications based on Web Services (WS) [WS], such as BPEL4WS [BPEL] and BPML [BPML], have emerged.

According to [BPEL], BPEL4WS allows specifying business processes and how they relate to Web Services which are described by e.g. WSDL [WSDL]. This includes specifying how a business process makes use of Web Services to achieve its goal, as well as specifying Web Services that are provided by a business process. Business processes specified in BPEL are fully executable and portable between BPEL-conformant environments. A BPEL business process interoperates with the Web Services of its partners, whether or not these Web Services are implemented based on BPEL. Finally, BPEL supports the specification of business protocols between partners and views on complex internal business processes.

As it is described in [BPML], BPML provides an abstracted execution model for collaborative and transactional business processes, and considers e-Business processes as made of a common public interface and as many private implementations as process participants. The execution model of BPML is based on a mathematical language that uses the pi-calculus model. BPML represents business processes as the interleaving of control flow, data flow, and event flow, while adding orthogonal design capabilities for business rules, security roles, and transaction contexts. BPML also offers explicit support for synchronous and asynchronous distributed transactions, and therefore can be used as an execution model for embedding existing applications within e-Business processes as process components. BPML is accompanied by a BPMN – Business Process Modelling Notation.

According to [Smith03], BPML provides unification of data, computation, and interaction which has the same flavor as the modelling technique supporting multiple perspectives suggested by us. In general, with regard to standard proposals, such as BPEL4WS and BPML, we agree with the statement that has been made in [Aalst03b], "Although there are well-established process modelling techniques combining expressiveness, simplicity and formal semantics (cf. Petri nets and process algebras); the software industry has chosen to ignore these techniques. As a result, the world is confronted with too many standards which are mainly driven by concrete products and/or commercial interests".

The UN/CEFACT Modelling Methodology (UMM) presented in [UMM] is an incremental business process and information model construction methodology that intends to create business documents that are exchanged between business partners based on business process models. UMM is affiliated to the ebXML (Electronic Business using eXtensible Markup Language) initiative [ebXML]. According to the UMM Meta Model, which defines the UMM modelling language, a commercial trading agreement is modelled as a business collaboration model. The UMM Meta Model is defined as an extension of the UML Meta Model by extending the UML stereotype syntax and semantics with the syntax and semantics of the business collaboration domain.

In [UMM], business processes are modelled as "business process use cases" which are refined into activity diagrams of UML. In this sense, UMM is similar to the Business Agents' Approach as we will see in section 3. However, agents and objects are not distinguished between in UMM. With regard to using UML activity diagrams, we agree with [EDO99] where it is argued that UML activity diagrams are more suitable for modelling computation processes than for business modelling, and that business process semantics needs another kind of behaviour specification.

Object-oriented Process, Environment, and Notation (OPEN) [OPEN] is a methodological approach that was designed for the development of software intensive applications, and particularly for the design and implementation of object-oriented and component-based software. OPEN was created and is maintained by the non-profit OPEN Consortium, an international group of over 35 methodologists, academics, CASE tool vendors and developers. OPEN provides strong support for the full lifecycle of a software application, including business process modelling.

According to [OPEN], OPEN is defined as a process framework, known as the OPF (OPEN Process Framework). This is a process metamodel from which can be generated an organization-specific process (instance). The metaclasses in the OPF are divided into five groups: Work Units, Work Products, Producers, Stages, and Languages. These form a component library for OPEN from which individual instances are selected and put together, in a constructor set fashion, to create a specific instance of OPEN.

OPEN includes the modelling notion of agent in the form of Producer. OPEN also distinguishes between human agents, agent roles, and institutional agents. Consequently, OPEN can in principle be used in agent-oriented modelling. This is reflected by the latest extension of OPEN to support agent-

oriented software development approaches which are reported about in [OPEN]. It is emphasized in [OPEN] that a modelling notation of one's choice can be used to document the work products, e.g. information systems, that the OPEN process produces. All this implies that the extended AORML diagrams could be incorporated into OPEN.

In the works such as [Dignum95] and [Weigand97], deontic logic is applied to the modelling of communication between autonomous cooperative systems and business process modelling. Deontic logic of obligation described e.g. in [Balzer00] and [Dignum99] offers notions such as responsibilities (obligations, duties) and authorizations (rights) that are attached to roles played by agents. Responsibilities of an agent that are directed towards other agents are termed 'commitments' in deontic logic.

In the paper [Dignum95], firstly the logical language for modelling communication based on speech acts and deontic logic is defined. The language is illustrated by an example about ordering products where each of the message types between the customer and the company is modelled with a logical formula. Thereafter it is shown how the formulas describing the exchange between the customer and the company can be represented in the formal specification language CoLa. According to [Dignum95], the communication protocols that are specified in CoLa are independent from the applications, but in contrast to traditional communication protocols, they capture the complete communication logic, not just an ordered set of messages.

Some elements of deontic logic are already used in AORML, based on [Wagner03a] where the first sketch of the deontic logic of AOR modelling is made. The main shortcomings of deontic logic' approaches from the perspective of information systems are that they are either of more philosophical nature like the one presented in [Balzer00], or, as it is stated in [Dignum97], "lack axiomatization or even a set of inference rules", and are therefore hard to use in practice.

The work reported on in [Barbuceanu99] concentrates on the interaction aspects of agents in the domain of integrated supply chain management, and particularly on the agents' mutual obligations and interdictions. It enables to define agent roles with the obligations, interdictions, and permissions attached to them. Such roles can then be used for e.g. coordinating the behaviours of the agents. The models in [Barbuceanu99] are, however, at a rather low level of abstraction and do not include models of information/knowledge possessed by agents.

The Action Workflow approach [Medina-Mora92] involves commitments in loops representing a four-step exchange between a customer and a performer. The main shortcomings of this approach are that it considers only two actors at a time and cannot easily model "run-time" modifications of the commitments.

According to [Karageorgos02], methodologies for engineering multi-agent systems can be divided into the following categories: *methodologies aligned with object-oriented software engineering*, *extensions to knowledge engineering methodologies*, *methodologies based on information systems methodologies*, and *methodologies highly coupled with specific agent-based system building toolkits*. Within the context of this thesis, we are naturally mostly interested in the methodologies based on information systems engineering. Out of them, Tropos [Mylopoulos01] has been reviewed in sections 2.1.4.2 and 2.1.4.3. We will now briefly describe two other methodologies which can be regarded as based on information systems engineering.

Firstly, the work [Elammari99] describes a design methodology which allows the development of agent based systems from user requirements. The models produced by the modelling approach are high-level model of the system, internal agent models, agent relationship models, conversational models, and contract models. The methodology provides a means of both visualizing the behaviour of systems of agents based on agent roles and contracts between them, and defining how the behaviour is achieved. It also includes an approach for generating from high-level designs implementable system definitions. The methodology does not, however, provide a graphical modelling language of its own. The approach has been applied to an intranet telephony application.

Secondly, in [Kendall96] a methodology for engineering agent-based systems is outlined. The methodology is based on the IDEF approach for workflow modelling and analysis [IDEF], the CIMOSA enterprise modelling framework [AMICE93], and the use case driven approach to object oriented software engineering. With the methodology proposed, agents can be identified along with their plans, goals, beliefs, sensors, and effectors based on function models represented using IDEF. The methodology also addresses the modelling of agent collaboration through scripts based on use cases and use case abstraction. The methodology does not, however, include explicit models of

information/knowledge possessed by agents. The methodology is illustrated by using a case study of discrete parts' manufacturing.

The UML Profile for Enterprise Distributed Object Computing [EDO99] defines a set of new modelling elements and relationships between them for the modelling of business processes. These constructs are based on the existing UML modelling concepts. It is argued that the modelling of the behaviour of business roles by business rules represents an alternative and/or complementary way of modelling the enterprise.

In [Odell00], an agent-oriented extension of UML, called AUML, has been proposed. It mainly concerns the expressivity of UML sequence and activity diagrams. However, AUML does not distinguish between agents and objects. In fact, UML class diagrams are not modified at all in AUML. Neither does it provide any support for business rules and business process modelling.

The RAMASD method described in [Karageorgos02] is a method for semiautomatic design of agent organizations based on the concept of role models as first-class design constructs. Role models represent agent behaviour, and the design of the agent system is done by systematically allocating roles to agents. Simple types of business rules can also be specified for roles. The core of the method is a formal model of basic relations between roles, termed role algebra. RAMASD thus focuses on the organizational view of agent-oriented modelling.

The informational view of agent-oriented modelling has a central position in the DESIRE methodology [Brazier97], where concepts and relations between them are defined in hierarchies for modelling multi-agent systems. That approach also includes rules that are allegedly used for automatic generation of prototype agent applications directly from their specifications.

3. DESCRIPTION OF THE BUSINESS AGENTS' APPROACH

3.1. THE CASE STUDY OF A CAR RENTAL COMPANY

For describing our approach, we will be making use of the case study of a fictitious car rental company by Model Systems, Ltd. called EU-Rent that is described in [BR00].

EU-Rent is a car rental company owned by EU-Corporation. EU-Rent has 1000 branches in towns in several countries. At each branch cars, classified by car group, are available for rental. Each branch has a manager and booking clerks who handle rentals.

Most rentals are by advance reservation; the rental period and the car group are specified at the time of reservation. EU-Rent will also accept immediate ('walk-in') rentals, if cars are available.

At the end of each day cars are assigned to reservations for the following day. If more cars have been requested than are available in a group at a branch, the branch manager may ask other branches if they have cars they can transfer to him/her.

Cars rented from one branch of EU-Rent may be returned to a different branch. The renting branch must ensure that the car has been returned to some branch at the end of the rental period. If a car is returned to a branch other than the one that rented it, ownership of the car is assigned to the new branch.

EU-Rent also has service stations, each serving several branches. Cars may be booked for maintenance at any time provided that the service station has capacity on the day in question. For simplicity, only one booking per car per day is allowed. A rental or service may cover several days.

A customer can have several reservations but only one car rented at a time. EU-Rent keeps records of customers, their rentals and bad experiences such as late return, problems with payment, and damage to cars. This information is used to decide whether to approve a rental.

In Figure 3-1, a simplified version of the organizational structure of the EU-Rent car rental company is depicted, showing only three branches, the headquarters, and just one automotive service station, serving the branches.

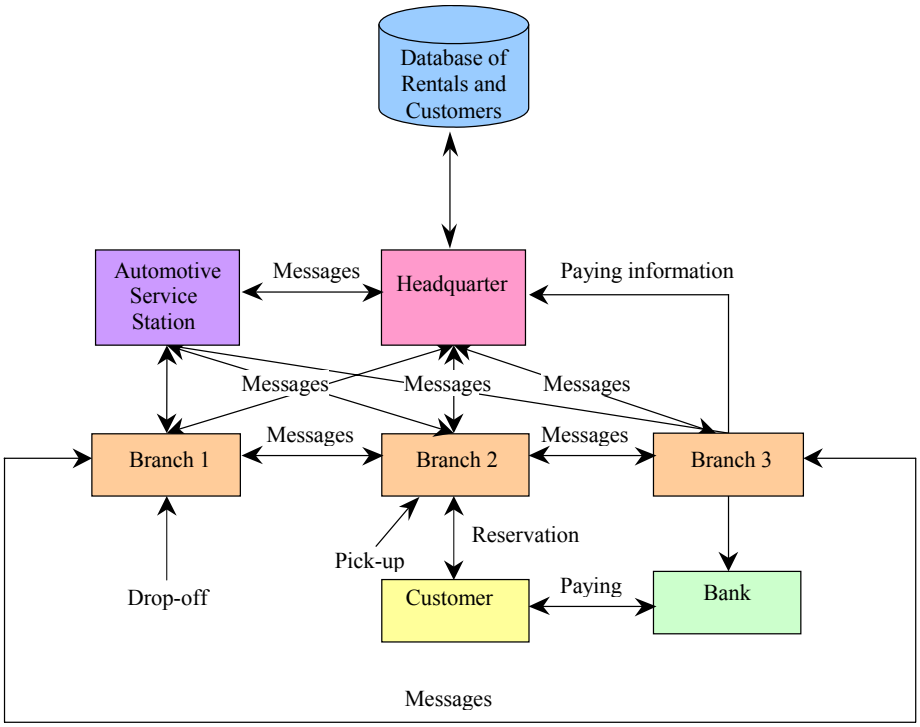


Figure 3-1. The EU-Rent car rental company

3.2. LEVELS OF BUSINESS MODELLING

Business rules have a *global nature*, i. e. they possibly involve objects of several object types. This doesn't fit into the principle of encapsulation that we have in object-oriented modelling. For example, the rule "product of the type A should never be cheaper than product of the type B" involves two different object types: it cannot be expressed within just one type. The rule "when the payment of a bill is two weeks overdue, it is required to send a reminder to the customer" involves several object types, an action, and time, and cannot therefore be encapsulated within one specific object type [Høydalsvik93].

There have been proposals to express business rules in an object-oriented fashion by using metamodelling in [Blanchard95] and [Odell95], but we are not aware of their any further applications. Many of the business modelling techniques that were described in Chapter 2 enable to model global business rules in a natural way. But in these techniques business rules are not properly connected to either actors (Ross Notation, Eriksson-Penker Extensions to UML, CIMOSA, BROCOM) or actions/activities (Eriksson-Penker Extensions to UML).

In [Metsker97], it is claimed that new, ontologically-oriented modelling and programming languages are needed that would allow "thinking about objects". In particular, according to [Metsker97], event though behaviours often occur as transactions, such as passing money between accounts, handing off material from a robot to an input conveyor, and lancing a boil, transaction processing is well understood, but barely supported in today's languages. An ontologically-oriented language will support the notion of transactions as structures of behaviour [Metsker97]. We are of the opinion that such transactions can be represented using business rules.

Conceptually, our solution to the problem of representing global business rules is adding the ***Agent Layer*** to the top of the Object Layer, like is shown in Figure 3-2. We understand the ***Object Layer*** in a wide sense of the term as either a relational, object-relational, or object-oriented database, Enterprise Resource Planning (ERP) or Enterprise Application Integration (EAI) system, or some object-oriented framework such as COM™ or CORBA™.

Agents of the Agent Layer communicate with each other by exchanging high-level typed messages, such as "ASK", "TELL", "REQUEST", and "PROPOSE". Following [Oja01], the communication between an agent on the Agent Layer and an object on the Object Layer is defined as a ***manipulation***. This term was coined to express the fact that objects are submitted to agents. Each agent has an ***object scope*** consisting of all object types whose instances are manipulated by it [Oja01].

We thus view data as agents' beliefs, and express business rules on the Agent Layer in terms of agents' beliefs and actions. This constitutes a powerful paradigm for the modelling, design, and implementation of business information systems.

Different modelling techniques like Gaia, which was described in section 2.2.3, can be applied to the modelling of agents understood this way. However, most of these modelling techniques, including Gaia, do not include information (knowledge) models. The only exception seems to be Agent-Object-Relationship (AOR) modelling proposed in [Wagner00a], [Wagner01], and [Wagner03a]. In this thesis, we make use of and extend the AOR Modelling Language (AORML) in combination with the Object Constraint Language (OCL) which is a part of the UML standard described in [OMG03a]. This combination of modelling techniques enables full-scale modelling and simulation of business processes based on business rules.

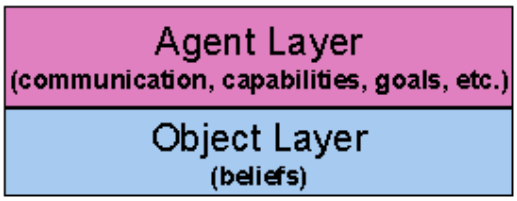


Figure 3-2. Adding the Agent Layer to the top of the Object Layer.

3.3. THE METAMODEL OF THE BUSINESS AGENTS' APPROACH

The metamodel of the Business Agents' Approach reflects the organizational, functional, motivational, informational, interactional, and behavioural views of agent-oriented modelling which were defined in section 1.5.6. It thus serves as a general metamodel of agent-oriented modelling. The metamodel depicted in Figure 3-3 also shows how the views of agent-oriented modelling are conceptually related to each other. All the entities in Figure 3-3 are *types* like in any object class model.

3.3.1. Organization Modelling

Following the definition presented in [Kieser92], we consider an ***organization*** as a social unit which lastingly strives to achieve common organization goals, as is reflected by Figure 3-3, and has a formal structure which coordinates the activities of all its members in order to achieve the goals. As Figure 3-3 shows, an organization consists of one or more functional organization units. A ***functional organization unit*** in Figure 3-3 (we term it simply *organization unit* hereafter) can be defined as an entity for managing the performance of activities to achieve one or more business goals of the organization [Uschold98]. Each organization unit maintains knowledge about a certain *functional subfield* of the problem domain that has *common business rules and goals* [Farhoodi96]. In our example case study of a car rental company, such organization units are the *branch*, *headquarters*, and *automotive service station*.

Business rules that an organization unit is responsible for and organization goals supported by them are attached to the organization unit through a number of its *internal* ***agents*** which can be biological agents (humans), artificial agents or other institutional agents. As Figure 3-3 reflects, an internal agent of an organization or organization unit can also be a *role*. A ***role*** is an abstract characterization of the behaviour of a social actor within some specialized context or domain of endeavor [Yu95a]. In [Zambonelli01], the following definition of a role is provided: "The role is what the agent is expected to do in the organization: both in cooperation with the other agents and in respect to the organization itself". Analogously, Curtis et al define a role as a coherent set of process elements to be assigned to an agent as a unit of functional responsibility [Curtis92]. In the example of a car rental company, the agent type *branch* includes the institutional roles *pick-up branch* and *drop-off branch*.

As Figure 3-3 reflects, organization and organization unit form subtypes of ***institutional agent***. According to Figure 3-3, an institutional agent may, in turn, include one or more roles. Both institutional agent and role are subtypes of ***agent***.

3.3.2. Function and Motivation Modelling

In the metamodel of Figure 3-3, each agent may include one or more prototypical job functions – *activities* – in an organization. The type of an ***activity*** (*task* in [Yu95a]) specifies a particular way of doing something [Yu95a]. For example, in a car rental company, activities of the types "Manage car reservation" and "Manage pick-up" are included by the role *pick-up branch*. An activity may consist of subactivities like an activity of the type "Manage car reservation" consists of subactivities of the types "Check the customer for blacklistedness", "Create rental reservation", and "Allocate a car".

As Figure 3-3 shows, an activity is started by an ***activity starting action*** which is invoked by a reaction rule in response to perceiving an event which can be a ***communicative*** or ***non-communicative*** (i.e. a physical) ***action event*** (i.e. an event that is created by an action) by other agent or a non-action event, particularly an *end of activity event* which is associated with the end of the previous activity.

According to Figure 3-3, an activity *may* be associated with an *agent goal*. An ***agent goal*** is a condition or state of affairs in the world that the agent would like to achieve [Yu95a]. As Figure 3-3 reflects, an agent goal is expressed in terms of *domain predicates – informational entity types*, *relationship types*, and *attributes*, and predicates of the problem domain defined by *derivation rules*. In order to achieve its goal, an agent performs an activity associated with that goal. In the example of car rental, a pick-up branch performs an activity of the type "Allocate a car" in order to achieve its agent goal which is expressed informally as "A car is allocated to the rental order".

As Figure 3-3 reflects, in function models an activity can be associated with one or more ***epistemic actions*** and/or *action events* (i.e. actions that are perceived by other agent(s) as events) performed by the corresponding agent. An ***action*** is an atomic unit of work done by an agent. As in [Shoham93], we view an agent's action in a broader sense as something that the agent *does* while e.g. in the "triangle" model of Figure 1-1 an action is understood narrowly as something that changes the state of a data object.

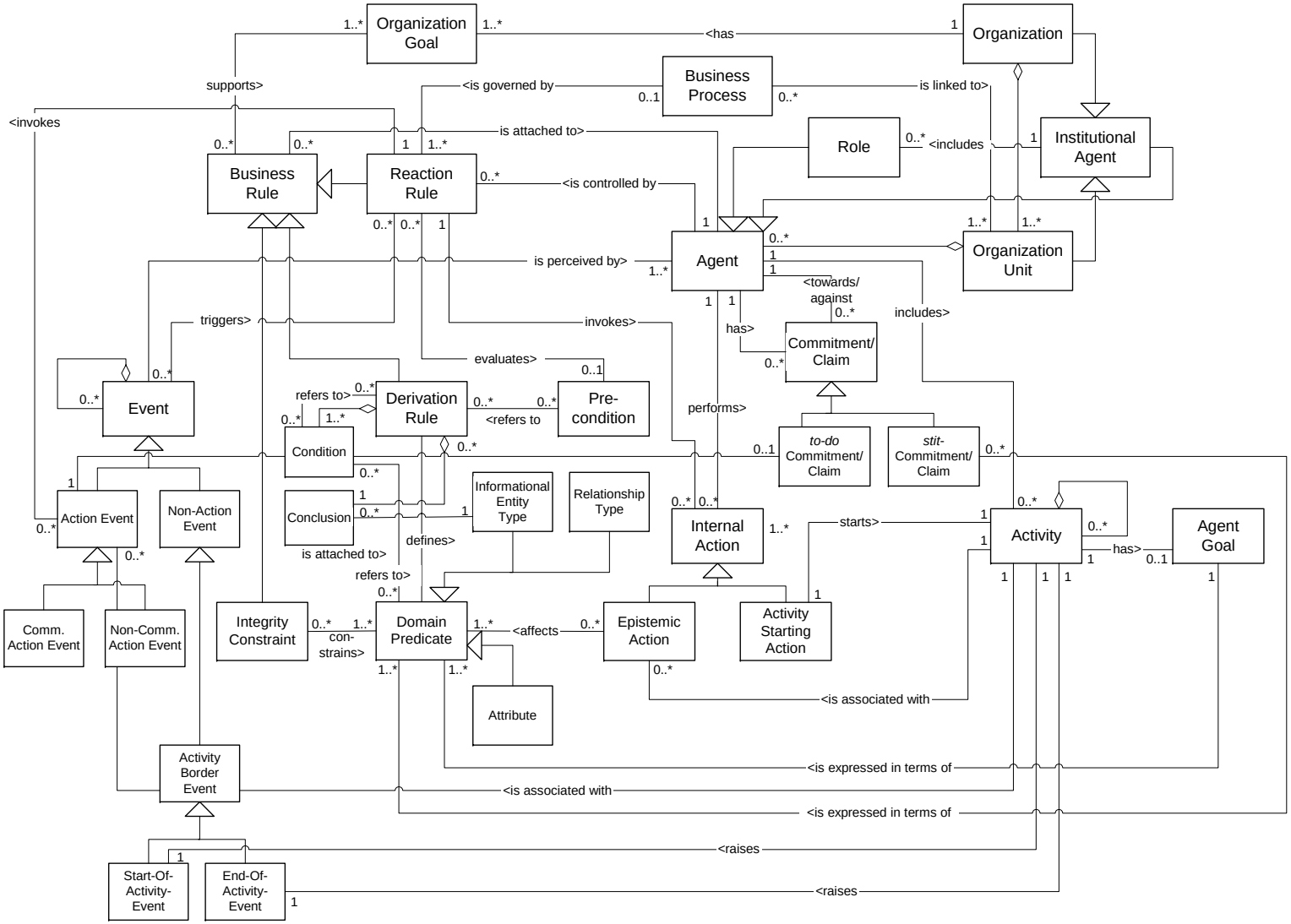


Figure 3-3. The metamodel of the Business Agents' Approach.

Actions performed by an agent can be divided into ***epistemic***, ***communicative***, and ***physical*** actions. Example actions of the respective three types in the car rental company are "Create an instance of RentalOrder with the status isPreliminary", "Ask another agent to transfer a car", and "Pick up the car".

3.3.3. Information Modelling

As Figure 3-3 reflects, an epistemic action affects one or more ***domain predicates***. There are three kinds of domain predicates: ***informational entity types***, ***relationship types***, and ***attributes***. ***Informational entity types*** are object types and representations of other agent types within agents. According to Figure 3-3, additional domain predicates may be defined by ***derivation rules*** which will be treated in section 3.8.2.1. Figure 3-3 also shows that a derivation rule consists of one or more ***conditions***, each of which may refer to other derivation rules, and a ***conclusion*** about an instance of the informational entity type appearing in the conclusion. For example, a derivation rule about an instance of the object type RentalCar states: "A car is available for rental (conclusion) if it is physically present, is not assigned to any rental order, is not scheduled for service, and does not require service". The conditions of this derivation rule refer to the object type RentalOrder and other derivation rules – status predicates isPresent, isScheduledForService, and requiresService of RentalCar.

Instances of domain predicates may be constrained by one or more ***integrity constraints***, as is shown in Figure 3-3. An example of an integrity constraint is "A customer of the car rental company must be at least 25 years old".

3.3.4. Interaction Modelling

Agents interact with each other. Communicative and physical actions of one agent are respectively perceived as ***communicative*** and ***non-communicative action events*** (i.e. events that are created by actions) by other agent(s). An agent may have one or more ***commitments*** towards and ***claims*** against other agents. In the metamodel presented in Figure 3-3, commitments and claims are subsumed under the notion ***commitment/claim*** because a commitment of one agent towards another agent is seen as a claim by the latter and the other way round. There are two kinds of commitments: commitments to perform actions of certain types, such as a commitment of one branch of the car rental company towards another to transfer a car, and *see-to-it-that* commitments to see to it that some domain predicate holds, such as a commitment to create a rental reservation. An action event may be coupled with a *to-do*-commitment. Analogously, there are ***claims*** against other agents that actions of certain kinds will be performed, such as a claim against a customer that he/she will pay for the rental, and claims to see to it that some condition holds, such as a claim to have the car serviced.

3.3.5. Behaviour Modelling

According to Figure 3-3, an event may trigger one or more reaction rules. ***Reaction rules*** were defined in section 1.4.1.1 as kinds of business rules that are concerned with the invocation and sequencing of actions and/or activities in response to events. As Figure 3-3 shows, after a reaction rule has been triggered by one or more triggering events, each of which possibly consists of other events, it may evaluate a precondition, which possibly refers to one or more derivation rules. Then a reaction rule may invoke one or more *internal actions* and/or one or more *action events*. An ***internal action*** is either an *epistemic action* or an *activity starting action*. An example of a reaction rule from the domain of car rental is: "When receiving from a customer the request to reserve a car of some specified car group (***triggering action event***), and there is enough capacity in the requested car group in the pick-up branch on the pick-up day (***precondition*** referring to the derivation rule about an instance of the object type CarGroup), the branch checks the customer for blacklistedness (***starting an activity*** *consisting of a set of actions*)".

There are two kinds of implicit ***activity border events*** which form special types of non-action events: a ***start of activity event*** and an ***end of activity event*** which are respectively associated with the start and end of an activity. As Figure 3-3 shows, performing an activity raises exactly one event of both kinds.

Figure 3-3 shows that a ***business process*** is governed by one or more reaction rules and linked to one or more organization units.

According to Figure 3-3, derivation rules, integrity constraints, and reaction rules form subtypes of ***business rules***. Figure 3-3 also reflects that a business rule is always attached to an agent and each business rule supports one or more organization goals.

3.4. OVERVIEW OF THE AGENT-OBJECT-RELATIONSHIP (AOR) MODELLING

In this section we describe, by using the example of car rental (v. section 3.1), how Agent-Object-Relationship (AOR) diagrams can be applied to the business modelling at two levels like it was suggested in section 3.2. The AOR diagrams were proposed in [Wagner00a], [Wagner01], and [Wagner03a] as an agent-oriented extension of Entity-Relationship-style or UML-style class diagrams. AOR modelling suggests that the semantics of business transactions can be more adequately captured if the specific *business agents* associated with the involved events and actions are explicitly represented in organizational information systems in addition to passive *business objects*. While both objects and agents are represented in the system, only agents *interact* with it, and the possible interactions may have to be represented in the system as well.

According to [Wagner03a], in AOR modelling, an entity is either an *agent*, an *event*, an *action*, a *claim*, a *commitment*, or an ordinary *object*. Only agents can communicate, perceive, act, make commitments and satisfy claims. Objects do not communicate, cannot perceive anything, are unable to act, and do not have any commitments or claims. Being entities, agents and objects share a number of attributes representing their properties or characteristics. So, in AOR modelling, there are the same notions as in ER modelling (such as entity types, relationship types, attributes, etc.).

Sections 3.4.1, 3.4.2, 3.4.3, and 3.4.4 are based on [Wagner03a] with the exception of the subsection 3.4.4.1 which is based on [Taveter01c]. Section 3.4.5 is largely based on [Wagner02].

3.4.1. Object and Agent Types

According to [Wagner03a], object types, such as sales orders or product items, are visualized as rectangles essentially in the same way like entity types in ER diagrams, or object classes in UML class diagrams. They may participate in *association*, *generalization*, or *aggregation/composition* relationships with other object types, and in *association* or *aggregation/composition* relationships with agent types.

Association types are represented by connection lines. The multiplicity constraints of an association are specified like in the UML (by means of declarations such as 0..1 or 1..* at the respective association end).

It is distinguished between *biological* agents, *institutional* agents, and *artificial* agents. For our purposes, humans form the only relevant subclass of biological agents. Examples of human agent types are Person, Employee, Student, Nurse, or Patient. Examples of institutional agents are organizations, such as a bank or a hospital, or organization units.

In certain application domains, there may also be artificial agent types, such as software agents (e.g., involved in electronic commerce transactions), embedded systems (such as automated teller machines), or robots. For instance, in an automated contract negotiation or in an automated purchase decision, a legal entity may be represented by an artificial agent. Typically, an artificial agent is owned, and is run, by a legal entity that is responsible for its actions.

In AOR diagrams, an agent type is visualized as a rectangle with rounded corners. Icons indicating a single human, a group, or a robot may be used for visualizing the distinction between human, institutional, and artificial agent. An agent type may be defined as a subclass of another agent type, thus inheriting all of its attributes (and operations). For example, Employee of EU-Rent is a subclass of Person.

Agents may be related to other entities by means of ordinary domain relationships (associations). In addition to the designated relationship types *generalization* and *composition* of ER/OO modelling, there are further designated relationship types relating agents with events, actions and commitments. They are discussed below.

An ***organization*** is viewed as a complex institutional agent defining the rights and duties of its internal agents that act on behalf of it, being involved in a number of interactions with *external agents*. ***Internal agents*** may be humans, artificial agents (such as software agents, agent information systems, robots or agent embedded systems), or institutional agents (such as organization units). An institutional agent consists of a number of internal agents that perceive events and perform actions on behalf of it, by playing certain *roles*. Internal agents, by virtue of their contractual status (or ownership status, in the case of artificial internal agents), have certain rights and duties, and assume a certain position within the subordination hierarchy of the institution they belong to.

As in the UML, instances of a type are graphically rendered by a respective rectangle with the underlined name of the particular instance as its title, possibly followed by a colon and its type, like

EU-Rent: Organization where Organization is another agent type. The same notation for instances also applies to objects, actions/events, and commitments/claims.

3.4.2. Actions and Events

According to [Wagner03a], in a business domain, there are various types of actions performed by agents, and there are various types of state changes, including the progression of time, that occur in the environment of the agents. For an external observer, both actions and environmental state changes constitute events. In the internal perspective of an agent that acts in the business domain, only the actions of other agents count as events.

Actions create events, but not all events are created by actions. Those events that are created by actions, such as delivering a product to a customer, are called ***action events***. Examples of business events that are not created by actions are the fall of a particular stock value below a certain threshold, the sinking of a ship in a storm, or a timeout in an auction. Such events are called ***non-action events***.

We make a distinction between ***communicative*** and ***non-communicative*** actions and events. Many typical business events, such as receiving a purchase order or a sales quotation, are communicative events. Business communication may be viewed as ***asynchronous*** point-to-point message passing. The expressions *receiving a message* and *sending a message* may be considered to be synonyms of *perceiving a communicative event* and *performing a communicative action*.

As opposed to the low-level (and rather technical) concept of messages in object-oriented programming, AOR modelling assumes the high-level semantics of speech-act-based *Agent Communication Language (ACL)* messages (see [KQML, FIPA]).

3.4.3. Commitments and Claims

According to [Wagner03a], commitments and claims are fundamental components of business interaction processes. Consequently, a proper representation and handling of commitments and claims is vital for automating business processes.

Representing and processing commitments and claims in information systems explicitly helps to achieve coherent behaviour in (semi-)automated interaction processes. In [Singh99], the social dimension of coherent behaviour is emphasized, and commitments are treated as ternary relationships between two agents and a 'context group' they both belong to. For simplicity, we treat commitments as binary relationships between two agents.

Commitments to perform certain actions, or to see to it that certain conditions hold, typically arise from certain communication acts. For instance, sending a sales quotation to a customer commits the vendor to reserve adequate stocks of the quoted item for some time. Likewise, acknowledging a sales order implies the creation of a commitment to deliver the ordered items on or before the specified delivery date.

There are two kinds of commitments: commitments to do an action and commitments to see to it that some condition holds. The former are called *to-do* commitments, and the latter *see-to-it-that* commitments. Formally, a *to-do* commitment of agent a_1 towards agent a_2 may be expressed as a quadruple,

$$\langle a_1, a_2, \alpha(c_1, \ldots, c_n), \mathit{TimeSpec} \rangle$$

where α denotes an action type, $c_1, \ldots, c_n$ is a suitable list of parameters, and *TimeSpec* specifies, e.g. in the form of a deadline, the time constraints for the fulfilment of the commitment. A *see-to-it-that* commitment is expressed in the same form, but now $\alpha(c_1, \ldots, c_n)$ represents a proposition (logical sentence) instead of an action term. The AOR modelling includes only *to-do* commitments because they are more fundamental. However, since, as we will show in section 3.8.3.2, it is not possible to create adequate business models without *see-to-it-that* commitments, in section 3.8.3.2 we will extend the AOR modelling language with them.

Commitment and claim processing (that is, the *operational semantics* of commitments and claims) includes the following operations:

- the *creation* of a commitment/claim through the performance of certain actions or the occurrence of certain events,
- the *cancellation* of a commitment by the debtor,
- *waiving* a claim by the creditor (or *releasing* the debtor from the corresponding commitment),
- the *delegation* of a commitment by the debtor to another agent who becomes the new debtor,
- *assigning* a claim by the creditor to another agent who becomes the new creditor,

- *fulfilling* a commitment.

3.4.4. External AOR Models

According to [Wagner03a], in an external AOR model, we adopt the view of an external observer who is observing the (prototypical) agents and their interactions in the problem domain under consideration. Typically, an external AOR model has a *focus* that is an agent, or a group of agents, for which we would like to develop a state and behaviour model. In this external-observer-view, 'the world' (i.e., the application domain) consists of various types of

1. ***agents***,
2. communicative and non-communicative ***action events***,
3. ***non-action events***,
4. ***commitments/claims*** between two agent types,
5. ordinary ***objects***,
6. various ***designated relationships***, such as *sends* and *does*,
7. ordinary ***associations***.

In the view of an external observer, actions are also events, and commitments are also claims, exactly like two sides of the same coin. Therefore, an external AOR model contains, besides the agent and object types of interest, the action event types and commitment/claim types that are needed to describe the interaction between the focus agent(s) and the other types of agents. These meta-entity types of external AOR modelling are shown in Figure 3-4.

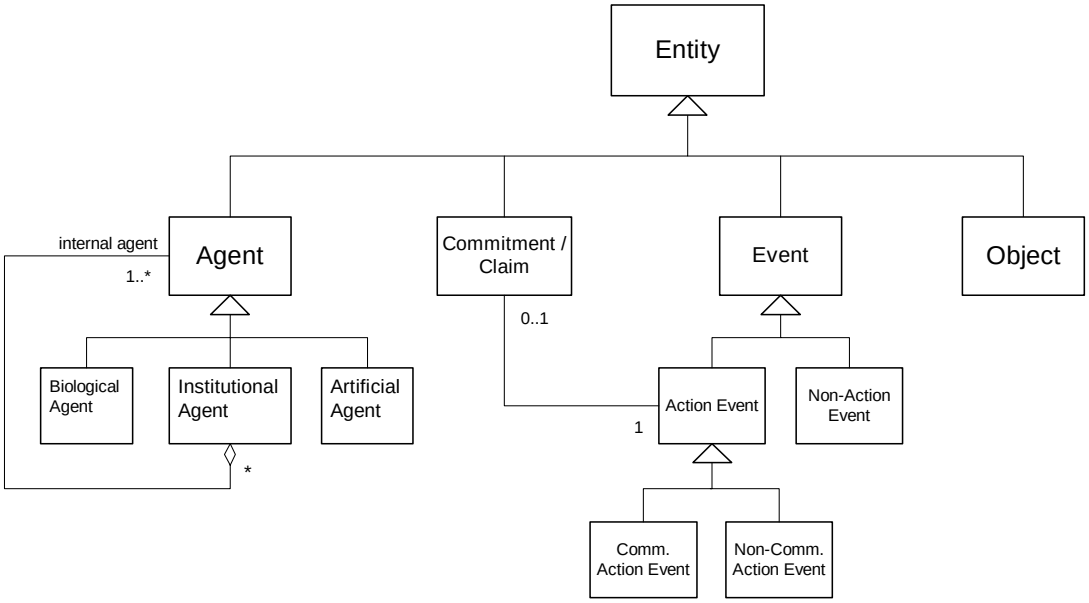


Figure 3-4. The meta-entity types of external AOR modelling.

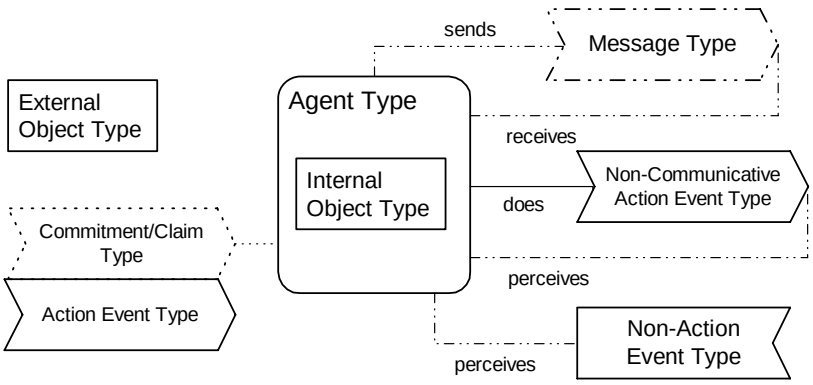


Figure 3-5. The core elements of external AOR modelling.

An external AOR model does not include any software artifacts. It rather represents a conceptual analysis view of the problem domain and may also contain elements which are merely descriptive and not executable by a computer program (as required for enterprise modelling).

The core elements of external AOR modelling are shown in Figure 3-5. An ***agent diagram*** of AOR modelling depicts the agents and agent types of a problem domain, together with their internal agents and agent types, their beliefs about objects and the relationships among them. The agent diagram of the domain of car rental is depicted in Figure 3-13.

According to [Wagner03a], in an external AOR model, the interactions between the focus agent(s) and the other types of agents are visualized in an ***interaction frame diagram***. In an interaction frame diagram, an action event type is graphically rendered by a special arrow rectangle where one side is an incoming arrow linked to the agent (or agent type) that performs this type of action, and the other side is an outgoing arrow linked to the agent (or agent type) that perceives this type of event. Communicative action event rectangles have a dot-dashed line. In the case of a non-action event, the corresponding event rectangle does not have an outgoing arrow (see Figure 3-6).

In an external AOR model, a *commitment* of agent a_1 towards agent a_2 to perform an action of a certain type (such as a commitment to return a car) can also be viewed as a *claim* of a_2 against a_1 that an action of that kind will be performed. Commitments/claims are conceptually coupled with the type of action event they refer to (such as provideCar action event in Figure 3-7). This is graphically rendered by an arrow rectangle with a dotted line on top of the action event rectangle it refers to, as depicted in Figure 3-6.

In an external AOR model, there are four types of designated relationships between agents and action events: sends and receives are relationship types that relate an agent with communicative action events, while does and perceives are relationship types that relate an agent with non-communicative action events. In addition, there are two types of designated relationships between agents and commitments/claims: hasCommitment and hasClaim. These designated relationship types are visualized with particular connector types as shown in Figure 3-6.

An interaction frame diagram, in an external AOR model, thus describes the possible interactions between two (types of) agents. It consists of various types of

1. communicative action events,
2. non-communicative action events,
3. commitments/claims (coupled with the corresponding types of action events), and
4. non-action events.

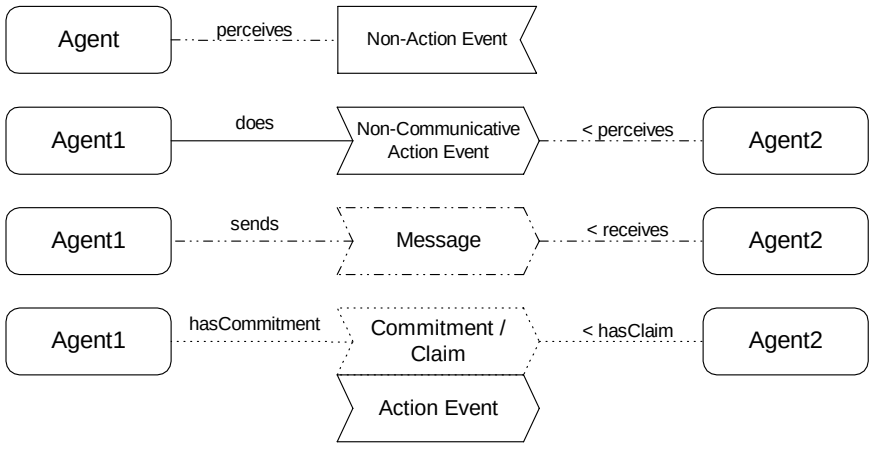


Figure 3-6. The designated relationship types sends, receives, does, perceives, hasCommitment, and hasClaim.

Figure 3-16 depicts the interaction frames between Customer and Branch, Branch and Headquarters, and Branch and AutomotiveServiceStation.

3.4.4.1. Reaction Rules and Interaction Pattern Diagrams

In [Taveter01c], we described the modelling of interaction process types by identifying interaction patterns and expressing them by means of ***reaction rules*** and ***interaction pattern diagrams***. Reaction rules may be used both for *describing* the reactive behaviour of all kinds of agents, and when possible, for the *executable specification* of the reaction patterns of an agent.

An example of a reaction rule is the following: *When a branch of the car rental company receives a request to reserve a car of some car group for a certain rental period from a customer, it first checks whether that car group has sufficient capacity during the rental period requested, and if this is the case, the branch sends a query to the headquarters to make sure that the customer is not blacklisted. Otherwise, the branch sends a refusal to the customer.* This rule is visualized as rule R1 in Figure 3-7.

A reaction rule is visualized as a circle with incoming and outgoing arrows drawn within the agent rectangle whose reaction pattern it represents. Each reaction rule has exactly one incoming arrow with a solid arrowhead: it represents the triggering event condition which is also responsible for instantiating the reaction rule (binding its variables to certain values). In addition, there may be ordinary incoming arrows representing conditions (referring to corresponding instances of other informational entity types) making up a precondition. There are two kinds of outgoing arrows. An outgoing arrow with an empty arrowhead denotes a mental effect referring to a change of beliefs and/or commitments. An outgoing connector to an action event type denotes the performance of an action of that type.

Reaction rules may also be represented in textual template form. For, instance, R1 and R3 could be expressed as in Table 3-1. In symbolic form, a reaction rule is defined as a quadruple

$$\varepsilon, C \rightarrow \alpha, F$$

where ε denotes the triggering event term, C denotes the precondition formula, α denotes the resulting action term, and F denotes the mental effect formula. Both C and F are formulas from a logical language corresponding to the (mental state) schema of the agent whose reaction pattern is specified by the rule [Wagner98]. It is required that all free variables in F occur also in C.

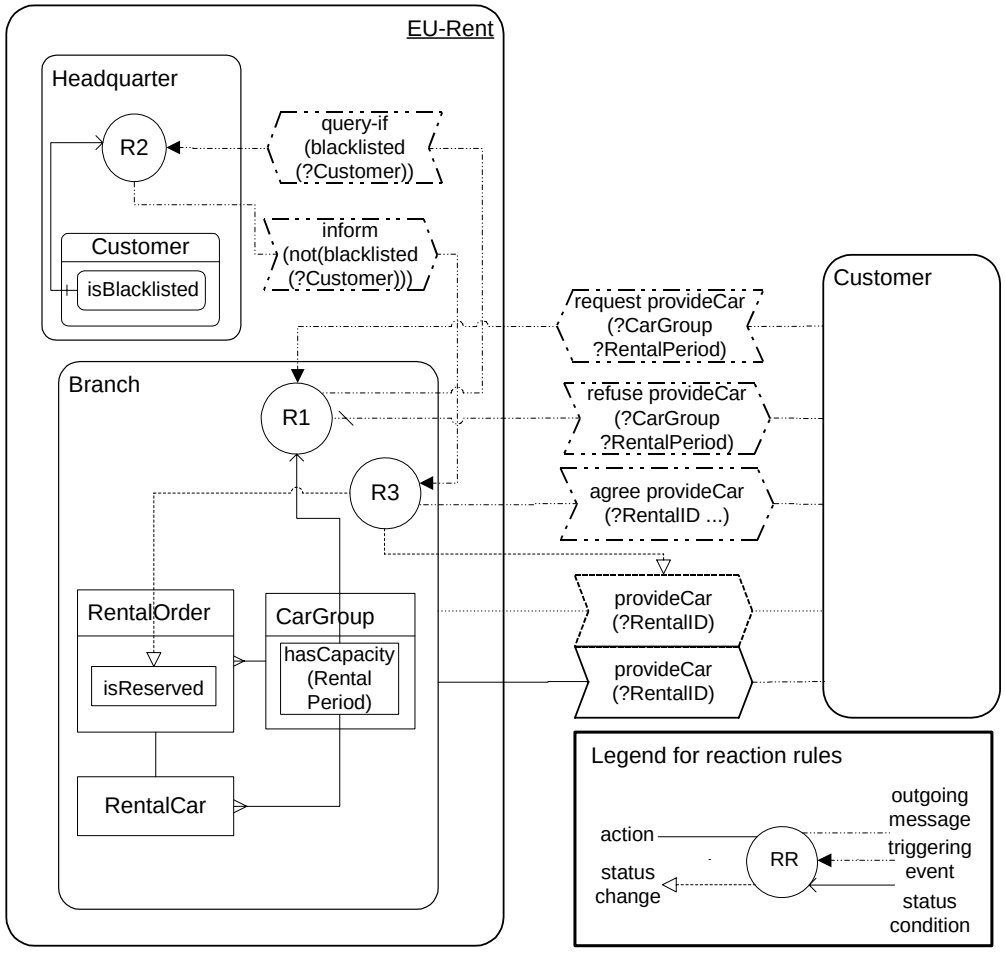


Figure 3-7. An interaction pattern diagram describing the process type where a customer requests a rental reservation from a branch of the car rental company EU-Rent, and where the capacity of the requested car group is first checked within the branch and after that the blacklistedness of the customer is checked with the headquarters. If there is enough capacity and the customer is not blacklisted, the rental reservation is created, and a confirmation is sent to the customer.

Table 3-1. The reaction rules R1 and R3 of Figure 3-7 in textual template form.

ON	Event	RECEIVE request provideCar(?CarGroup ?RentalPeriod) FROM ?Customer
IF	Condition	?CarGroup.hasCapacity(?RentalPeriod)
THEN	Action	SEND query-if(isBlacklisted(?Customer)) TO ?Headquarters
ELSE	Action	SEND refuse provideCar(?CarGroup ?RentalPeriod) TO ?Customer
ON	Event	RECEIVE inform(not(isBlacklisted(?Customer))) FROM ?Headquarters
THEN	Effect	COMPUTE ?RentalID = getNewRentalID() CREATE BELIEF RentalOrder(?RentalID, ?Customer, isPreliminary, . . .) CREATE COMMITMENT TOWARDS ?Customer TO provideCar(?RentalID) BY . . .
	Action	SEND agree provideCar(?RentalID) TO ?Customer

Notice that in an interaction pattern diagram, the *actions performed by one agent may be at the same time the events perceived by another agent*. An interaction pattern can therefore visualize the *reaction chains* that arise by one reaction triggering another one.

3.4.5. A UML Profile of the AOR Metamodel

The Agent-Object-Relationship Modelling Language (AORML), which was described in sections 3.4.1 through 3.4.4, can be viewed as an extension to UML [OMG03a]. It is stated in [Wagner02] that AORML, by virtue of its agent-oriented categorization of different classes, allows more adequate models of organizations and organizational information systems than plain UML.

In [Wagner02] the AOR metamodel is represented as a UML profile. According to [Wagner02], this allows AOR models to be notated using standard UML notation. This means that most UML tools (specifically the ones that support the extension mechanisms of UML, such as stereotypes and tagged values) can be used to define AOR models. Standard practice for defining UML profiles has been adopted. A mapping of AOR metamodel classes to their base UML classes, with accompanying stereotypes, tagged values, and constraints is presented. An implementation of this mapping can be used, for example, to generate XMI metadata conforming to the AOR metamodel from models notated using the UML profile. Specialized AOR tools will more likely directly use the AOR metamodel rather than the UML profile as a basis for storing and manipulating models [Wagner02].

The only notion of AORML that can not be represented by means of a UML profile is *reaction rule* because such a notion is not compatible with UML.

A summary of the stereotypes of external AOR modelling together with the extensions to be proposed by us in the following sections is presented in Table 3-2. The table has been adopted from [Wagner02] and extended by the additional stereotypes Activity, AutomaticActivity, HumanActivity, and SemiautomaticActivity to be explained in section 3.8.5.1. *Restricted generalization* means that whenever a generalization relationship involves a class of that stereotype as either subclass or superclass, the other class involved must also be of that stereotype. *No aggregation* means that classes of that stereotype must not participate in any aggregation.

As Table 3-2 shows, in addition to the stereotypes Agent and Object, also the stereotype ActionEvent along with its subclasses CommunicativeActionEvent and NonCommunicativeActionEvent extends the metaclass Class. According to [OMG03a], the attributes that are defined for a stereotype are instantiated for any UML class that the stereotype is applied to. In such a way, the value of the String-type attribute performative defined for the stereotype CommunicativeActionEvent specifies the performative type pertaining to a communicative action event type like "request".

Analogously, the subclass ToDoCommitmentClaim of the stereotype CommitmentClaim defines the attribute actionEventTypeName of the type String. When the stereotype is applied to a *to-do*-commitment/claim type, this attribute refers to the name of the action event type that the commitment/claim type is coupled with. Another subclass STITCommitmentClaim of the stereotype CommitmentClaim does not introduce any new attributes. It is applied to *stit*-commitment/claim types which are anonymous classes [OMG03b] distinguished by the types of propositions included by them.

Table 3-2. Stereotypes of the extended AOR modelling.

Stereotype	Base Class	Parent	Constraints
AORModel	Model	NA	
Agent	Class	NA	Restricted generalization.
BiologicalAgent	Class	Agent	Restricted generalization.
HumanAgent	Class	BiologicalAgent	Restricted generalization.
ArtificialAgent	Class	Agent	Restricted generalization.
SoftwareAgent	Class	ArtificialAgent	Restricted generalization.
Robot	Class	ArtificialAgent	Restricted generalization.
EmbeddedSystem	Class	ArtificialAgent	Restricted generalization.
InstitutionalAgent	Class	Agent	Restricted generalization.
Organization	Class	InstitutionalAgent	Restricted generalization.
OrganizationalUnit	Class	InstitutionalAgent	Restricted generalization.
Object	Class	NA	Restricted generalization.
Event	Class	NA	Restricted generalization. No aggregation.
ActionEvent	Class	Event	Restricted generalization. No aggregation.
Communicative Action Event	Class	Action Event	Restricted generalization. No aggregation.
NonCommunicative Action Event	Class	Action Event	Restricted generalization. No aggregation.
NonActionEvent	Class	Event	Restricted generalization. No aggregation.
CommitmentClaim	Class	NA	Restricted generalization. No aggregation.
ToDo CommitmentClaim	Class	CommitmentClaim	Restricted generalization. No aggregation.
STIT CommitmentClaim	Class	CommitmentClaim	Restricted generalization. No aggregation.
Activity	Class	NA	Restricted generalization. No aggregation.
AutomaticActivity	Class	Activity	Restricted generalization. No aggregation.
HumanActivity	Class	Activity	Restricted generalization. No aggregation.
Semiautomatic Activity	Class	Activity	Restricted generalization. No aggregation.
does	Association	NA	The domain class must be an agent type and the range class must be a non-communicative action event type. Multiplicity is one-to-many.
perceives	Association	NA	The domain class must be an agent type and the range class must be a non-communicative action event type or a non-action event type. Multiplicity is one-to-many.
sends	Association	NA	The domain class must be an agent type and the range class must be a communicative action event type. Multiplicity is one-to-many.
receives	Association	NA	The domain class must be an agent type and the range class must be a communicative action event type. Multiplicity is one-to-many.
hasClaim	Association	NA	The domain class must be an agent type and the range class must be a commitment/claim type. Multiplicity is one-to-many.
hasCommitment	Association	NA	The domain class must be an agent type and the range class must be a commitment/claim type. Multiplicity is one-to-many.
performs	Association	NA	The domain class must be an agent type and the range class must be an activity type. Multiplicity is one-to-many.

3.5. INCORPORATING THE OBJECT CONSTRAINT LANGUAGE

The Object Constraint Language (OCL), which is now a part of the UML standard [OMG03a], is a formal language used to express integrity constraints (called 'invariants') as well as preconditions and postconditions for operations on the basis of a vocabulary defined by a UML model.

OCL is a pure specification language; therefore, an OCL expression is guaranteed to be without side effect. When an OCL expression is evaluated, it simply returns a value. It cannot change anything in the model. This means that the state of the system will never change because of the evaluation of an OCL expression, even though an OCL expression can be used to *specify* a state change (e.g., in a post-condition).

OCL is a typed language, so that each OCL expression has a type. To be well formed, an OCL expression must conform to the type conformance rules of the language. For example, you cannot compare an Integer with a String. OCL includes a set of supplementary predefined types.

Each OCL expression is written in the context of an instance of a specific informational entity type (i.e. an object type or a representation of an agent type). In an OCL expression, the reserved word **self** is used to refer to the contextual instance. For instance, if the context is Customer, then **self** refers to an instance of Customer.

Within the Business Agents' Approach, OCL is used for the following purposes:

- To specify integrity constraints as invariants on informational entity types.
- To define derived attributes by means of OCL invariants.
- To define status predicates by means of modified OCL invariants.
- To define intensional predicates by means of operations.
- To specify input parameters of activities.
- To specify preconditions and goals (postconditions) of activities as logical expressions.
- To specify preconditions of reaction rules as logical expressions.
- To specify postconditions (mental effects) of reaction rules as logical expressions.
- As a navigation language.

For making possible the use of OCL for the purposes listed above, we will introduce a number of modifications and extension into it. For example, *we will extend OCL by allowing more than one contextual instance*.

If the constraint is shown in an agent diagram of AORML, with the proper stereotype and the dashed lines to connect it to its contextual element, there is no need for an explicit context declaration in the test of the constraint. The context declaration is optional.

Based on [OMG03a], starting from a specific object, we can navigate an association on the agent diagram to refer to other informational entities and their properties. To do so, we navigate the association in the direction of the rolename of the opposite association-end. The value of this expression is the set of informational entities on the other side of the association. If the multiplicity of the association-end has a maximum of one ("0..1" or "1"), then the value of this expression is an informational entity. By default, navigation will result in a Set of informational entities. When the association on the agent diagram is adorned with {ordered}, the navigation results in an ordered Sequence of informational entities. When a rolename is missing at one of the ends of an association, the name of the type at the association end, starting with a lowercase character, is used as the rolename. If this results in an ambiguity, the rolename is mandatory. In AORML, it is also possible to navigate from an internal agent or object type to the enclosing agent or object type.

Appendix A describes the grammar for the version of OCL that we use in combination with AOR modelling. The grammar is based on the definitions of OCL in [OMG03a] and [OMG03b]. It also includes a number of modifications and extensions to the standard OCL [OMG03a] that will be described in the corresponding sections. The core of OCL is formed of logical expressions which are used to specify invariants, status and intensional predicates, preconditions and goals (postconditions) of activities, and pre- and postconditions (mental effects) of reaction rules.

The grammar description in Appendix A uses the EBNF syntax, where "|" and "?" respectively stand for a choice and optionally, "*" means zero or more times, "+" means one or more times, and expressions delimited with "/*" and "*/" are definitions described with English words or sentences. In the description of *string,* the syntax for lexical tokens from the JavaCC parser generator is used.

In the sequel, we use the UML term "association" and the term "relationship" interchangeably.

3.6. EXTENDING AOR MODELLING BY ACTIVITY DIAGRAMS

3.6.1. Introduction of Activity Diagrams

Figure 3-7 in section 3.4.4.1 depicts an interaction pattern diagram describing a part of the business process type of car rental. Figure 3-7 demonstrates that an interaction pattern diagram can visualize the reaction chains that arise by one reaction triggering another one. However, *for adequate modelling of business processes interaction pattern diagrams are not sufficient because they do not enable to model action sequences*. For this reason, we need to introduce activities as a glue connecting the actions of an agent within a business process to each other.

In [Eshuis02b], an ***activity*** is defined using workflow terminology as an uninterruptible amount of work that is performed in a non-zero span of time by an actor. Each activity belongs to some *activity type*. An ***activity type*** (*task* in [Yu95a]), like "Manage car reservation", is defined as a prototypical job function in an organization which specifies a particular way of doing something [Yu95a]. It seems natural to allow specifying the start of an activity in the action part of a reaction rule. In other words, an instance of an activity type is created by means of a reaction rule in response to perceiving an event.

For graphical modelling of activity types, we are introducing an extension to AOR modelling – ***activity diagrams*** – which combine interaction frame diagrams and interaction pattern diagrams with the notion of activity. To enable the modelling of activities by activity diagrams, we are extending the set of possible types of action terms α of the reaction rule quadruple ε, $C \rightarrow \alpha$, F defined in section 3.4.4.1 by the START ACTIVITY activityReference construct type, representing the type of an ***activity starting action***, where activityReference denotes a reference to the activity type whose instance is started by the construct. We also allow both the triggering event term ε and the action term α to consist of specifications of more than one event and action types, respectively, connected with logical conjunction(s).

In activity diagrams, activity types are visualized as rectangles with rounded left and right sides, as is shown in Figure 3-8. An activity can be started by a reaction rule as is shown in Figure 3-8 a) where an activity of the type ActivityType1 is started in reaction to perceiving an action event of the type ActionEventType1. When an activity has been started by a reaction rule, the agent is in the corresponding ***activity state***. Using the workflow terminology described in [Eshuis02b], in an activity state an actor is executing an activity in an instance of a *case* (*business process* in AORML). Starting an activity is equal to creating an instance of the corresponding activity type. In the course of executing an activity, data relevant for the business process instance the activity is a part of is updated. In the extended AOR modelling, this data is represented as ***input parameters*** of an activity. Input parameters correspond to *case attributes* [Eshuis02b] in workflow terminology. The names and types of input parameters of an activity are represented in an activity diagram by using the enclosed in parentheses formalParameterList construct of OCL, which is defined in Appendix A. In the activity diagram of Figure 3-9, an activity of the type "Manage car reservation" is started by reaction rule R1. The parameters that are passed to the activity are specified in parenthesis following the activity name.

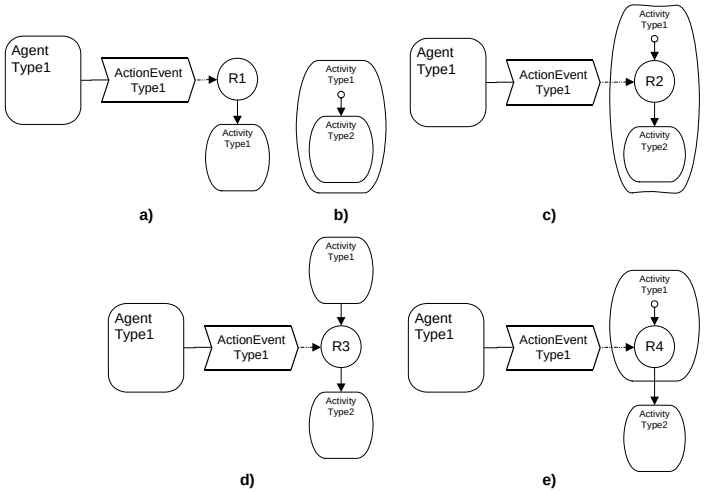


Figure 3-8. The constructs for starting of activities.

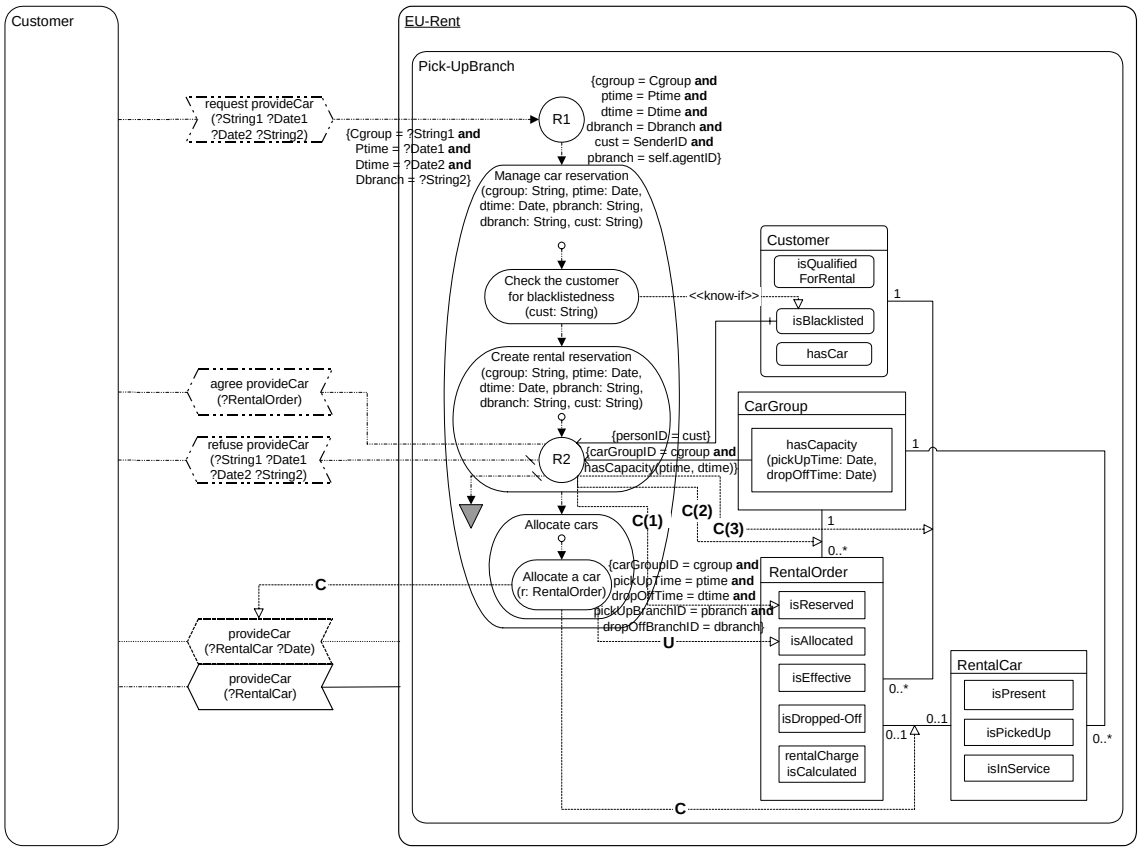


Figure 3-9. Activity type for the business process type of car reservation.

Each reaction rule is characterized by one or more *internal logical variables* which form the ***schema*** of the reaction rule. The number and types of the internal variables of a rule are determined by the number and types of the values that are assigned to them. When an activity is started by means of a reaction rule, to its input parameters are assigned the values of the internal variables of the invoking reaction rule. The internal variables, in turn, are instantiated by matching the event condition of the reaction rule with the triggering event instance(s). In such a case, the number and types of the internal variables are determined by the number and types of the data fields included by the triggering event(s). In Figure 3-9 is specified the evaluation of the internal variables Cgroup, Ptime, Dtime, Dbranch, and SenderID of reaction rule R1 and the assignment of their values to the corresponding input parameters cgroup, ptime, dtime, dbranch, and cust of an activity instance of the type "Manage car reservation". The name of an internal variable of a reaction rule in an activity diagram always starts with a capital letter, while the name of an activity's input parameter starts with a small letter.

If there are no two or more source values of the same type, the evaluation of internal variables of a reaction rule and the assignment of their values to e.g. the input parameters of an activity started by the reaction rule is implicit in activity diagrams. Otherwise, it should be explicitly specified. Internal variables of a reaction rule will be treated more thoroughly in section 3.6.3. The evaluation of the input parameters that do not have counterparts among internal variables, like the input parameter pbranch (the identifier of the pick-up branch) in Figure 3-9, should also be explicitly specified.

There are two activity border events implicitly associated with the beginning and end of each activity. Through an ***activity border event*** of the type START activityTemplate, defined in Appendix B, where activityTemplate includes the identifier of the activity type, an activity can trigger its subactivity or internal reaction rule. If an event of this kind triggers the activity's internal reaction rule, activityType may also include a list of internal variables to be evaluated by the event. The triggering event type START activityTemplate is graphically represented by an empty circle with the outgoing arrow to the symbol for the activity type or internal reaction rule. According to the pattern shown in Figure 3-8 b), upon the start of an activity of the type ActivityType1, its subactivity of the type ActivityType2 is started. In Figure 3-8 c), an activity border event of the same type triggers reaction rule R2 included by the activity. The construct depicted in Figure 3-8 c) allows to represent waiting for a message within an

activity: after the start of an activity of the type ActivityType1, if the agent perceives an action event of the type ActionEventType1, a subactivity of the type ActivityType2 is started.

Additionally, each activity is associated with another implicit activity border event of the type END activityID, defined in Appendix B, which can trigger a subsequent activity or reaction rule. This event type is visualized by drawing a triggering arrow from the activity type symbol to either the symbol of the next activity type or to the symbol of the reaction rule triggered by an activity of the corresponding type. The pattern shown in Figure 3-8 d) allows to represent waiting for a message between two activities by referring to the activity border event of the type END activityID. According to this pattern, after the end of an activity of the type ActivityType1, if the agent perceives an action event of the type ActionEventType1, an activity of the type ActivityType2 is started.

The pattern shown in Figure 3-8 e) allows to represent starting the next activity (which is not a subactivity of the previous one) upon perceiving an action event. In Figure 3-8 e), reaction rule R4 within an activity of the type ActivityType1 starts a subsequent activity of the type ActivityType2 only if it perceives an action event of the type ActionEventType1. As we will see in section 3.8.5.3, the pattern of starting the next activity upon perceiving an action event allows to represent the behavioural pattern "Deferred choice" which is generally hard to implement according to [Patterns03].

An activity may consist of subactivities. The activity types for which no subactivity types can or is desired to be identified are termed ***elementary activity types***. An elementary activity type, like the activity type "Create rental reservation" in Figure 3-9, can contain at most one reaction rule that is triggered by a START activityTemplate event. As any reaction rule, it can invoke a number of physical and/or communicative actions. We thus have actions as the basic elements of a business process. An *action* happens at a time point (i.e., it is immediate), while an *activity* is being performed during a time interval (i.e., it has duration), and consists of a set of actions. An activity type is ***completely/partially specified*** if all/some of its actions are specified by reaction rules. Otherwise, the activity type is ***unspecified.*** For example, the activity type "Create rental reservation" in Figure 3-9 is completely specified, while the activity type "Allocate cars" in the same figure is unspecified.

From the perspective of a particular agent, completely specified activities can be viewed as *transactions* because a completely specified activity can be characterized by the so-called ACID-properties [Gray93] which are paraphrased for elementary activities as follows:

- *atomicity*: all or none of an activity is performed;
- *consistency*: an activity preserves the consistency of the agent's VKB;
- *isolation*: intermediate results of an activity are not visible to any other activity or agent;
- *durability*: when an activity concludes successfully, its effects are permanent.

For a reaction rule included by an activity type, like for rule R2 in Figure 3-9, internal variables are determined by the input variables of the enclosing activity type and the data fields included by the triggering event(s). For example, reaction rule R2 in Figure 3-9 contains the internal variables Cgroup, Ptime, Dtime, Pbranch, Dbranch, and Cust that are all determined by the input parameters defined for the activity type "Create rental reservation". The schema of a reaction rule can also be complemented by the rule's precondition which will be treated in section 3.6.4.

An activity passes the names and values of its input parameters to the activities of the next level included by it. For example, in Figure 3-9 an activity of the type "Manage car reservation" passes the names of its input parameters cgroup, ptime, dtime, pbranch, dbranch, and cust and their values to the activity of the type "Create rental reservation" included by it. Definitions of these input parameters are therefore repeated for the activity type "Create rental reservation" in Figure 3-9. In the same way, an activity of the type "Manage car reservation" passes the name and value of its input parameter cust to the activity of the type "Check the customer for blacklistedness" started by it. An activity can thus access the input parameters that are defined for any enclosing activity type. Re-specifying input parameters for an enclosed activity type is not obligatory in an activity diagram.

3.6.2. Preconditions and Goals of Activities

Occurrence of the triggering event of a reaction rule, such as receiving of the request(provideCar((?String1 ?Date1 ?Date2 ?String2)))[6] agent message in Figure 3-9, or occurrence of an activity border event of the type END Create_rental_reservation in the same figure, is a *sufficient* condition for an activity instance to start. An activity may have other *necessary* conditions that must

[6] The parameters ?String1, ?Date1, ?Date2, and ?String2 respectively stand for the identifier of the pick-up branch, pick-up time, drop-off time, and the identifier of the drop-off branch.

also be true when it is started. They may refer to status or intensional predicates of informational entity types and can be defined by means of OCL. Such conditions constitute the ***precondition*** of an activity. For example, the precondition of an activity of the type "Allocate a car" in Figure 3-9 is the existence of the instance of RentalOrder which has the status isReserved, is to be picked up on the following day, and is referred to by the activity's input parameter r of the type RentalOrder. In OCL, this precondition is represented as follows:

```
RentalOrder.allInstances->exists(ro : RentalOrder | ro.isReserved and ro.pickUpTime.date = now().date + 1
and ro = r)
```

Each activity may also be characterized by its ***goal*** which is a condition or state of affairs in the world that the agent would like to achieve [Yu95a]. According to the terminology introduced in [Presley97], the goals modelled within the Business Agents' Approach are always *process goals*. A ***process goal*** is a special case of a goal which is tied to a specific activity. In this case, the *assignee* of the goal is the agent that includes the activity. For example, the assignee of the goal of an activity of the type "Manage car reservation" in Figure 3-9 is an institutional agent of the type Pick-Up Branch.

We distinguish between ***goal types***, where goals are propositions that possibly contain uninitialized variables, and ***goal instances***, where all variables are initialized. Goal types are attached to activity types which are, in turn, assigned to agent types, while instances of goals characterize activity instances which are performed by agent instances. For the sake of simplicity, we will subsequently use the term 'goal' for both goal types and instances.

Strictly speaking, there is a difference between goals and postconditions. For example, the creation of a commitment to provide the customer with the car should not really be a part of the agent's goal (since it is semantically implied by social norms), but it should be a part of the postcondition of the corresponding activity of the type "Allocate a car" shown in Figure 3-9. However, for the sake of simplicity of the Business Agents' Approach, our term 'goal' subsumes both goals and postconditions.

The precondition and goal are defined for an activity type in terms of input parameters of the activity type. For example, the goal of an activity of the type "Manage car reservation" presented below specifies that an instance of RentalOrder with the attribute values corresponding to the values of the activity's input parameters has been created, the *many-to-one* relationships between the RentalOrder created and the corresponding instances of CarGroup and Customer, that are identified by the values of the input parameters cgroup and cust, respectively, have been formed, a car has been allocated to the rental order (i.e. the *one-to-one* relationship has been created between the RentalOrder and the instance of RentalCar), the instance of RentalOrder created has the status isAllocated, and the *to-do*-commitment to provide the customer with the car by the pick-up time specified in the rental order has been created:

```
RentalOrder.allInstances->exists(r: RentalOrder | r.carGroupID = cgroup and
r.pickUpTime = ptime and r.dropOffTime = dtime and r.pickUpBranchID = pbranch and
r.dropOffBranchID = dbranch and r.carGroup->exists(cg : CarGroup | carGroupID = cgroup and
cg->includes(r)) and r.customer->exists(c : Customer | personID = cust and c->includes(r)) and
r.rentalCar->exists(c : RentalCar | c.rentalOrder = r) and
r.isAllocated and provideCar.allInstances->exists(about = r.rentalCar and
dueTime = ptime and sourceID = pbranch and targetID = cust))
```

The goal defined for an unspecified activity type is visualized by one or more mental effect arrow(s) leading from the activity type rectangle to the symbol(s) for the object and/or relationship type(s) that the goal is related to, like the goal defined for the activity type "Allocate a car" in Figure 3-9. For a completely specified activity type, like for the elementary activity type "Create rental reservation" in Figure 3-9, the goal is visualized as one or more mental effect arrows originating in the reaction rule symbol included by the activity type rectangle. A mental effect arrow of a completely specified activity type may be augmented by an OCL expression as will be described in section 3.6.5.

3.6.3. The Schema of a Reaction Rule

The internal variables of a reaction rule forming the ***schema*** of the rule can be treated as *logical variables* $x_1, ..., x_n$. According to [Sterling86], a ***logical variable*** stands for an unspecified but single entity, rather than for a store location in memory like a variable in a conventional programming language. The internal logical variables of a reaction rule are instantiated at rule application time by values of the following types:

- the values of the data items included by the triggering event(s);

- the values that can be retrieved from the agent's VKB[7] so that the reaction rule's precondition is satisfied in the current VKB state of the agent;
- the values of the input variables of the enclosing activity type.

For example, the internal variables within the scope of reaction rule R1 are Cgroup, Ptime, Dtime, Dbranch, and SenderID. As is expressed in Figure 3-9, the internal variables mentioned are instantiated so that their values are equal to the values of the data fields of the request(provideCar((?String1 ?Date1 ?Date2 ?String2))) agent message, and the value of the internal variable SenderID is equal to the identifier of the instance of Customer who sent the message. The latter is a "standard" internal variable to which is always assigned the identifier of the agent from which the triggering action event originates. A reaction rule that is triggered by a non-action event does not have the internal variable SenderID. As another example, to the internal variables Cgroup, Ptime, Dtime, Pbranch, Dbranch, and Cust of reaction rule R2 in Figure 3-9 are assigned the values of the respective input parameters of the enclosing activity of the type "Create rental reservation". In addition to the internal variables mentioned, the schema of reaction rule R2 includes the internal variables corresponding to the instance of Customer and instance of CarGroup that are retrieved from the agent's VKB. Since the mechanism of evaluating such internal variables is the same as the mechanism of variable binding in the Prolog programming language, we will next describe it by using examples presented in Prolog.

The internal variables of a reaction rule can be represented in Prolog as the respective relations or *predicates* such as customer and car_group. According to [Sterling86], predicates can be stated through *facts* like customer(1245). Predicates can also be defined by means of Prolog *rules* which are statements of the form $A \leftarrow B_1, B_2, \ldots, B_n$ where $n \geq 0$, A is the head of the rule, and the B_i's are the rule's body. The backward arrow $\leftarrow$ is used to denote logical implication. Variables appearing in rules are universally quantified, and their scope is the whole rule. In Prolog, from any universally quantified statement P, like a rule, an instance of it $P\theta$ can be deduced, for any variable substitution θ [Sterling86]. More precisely: if the body of a rule has free variables, a variable substitution is retrieved such that the rule's body becomes an inferable logical sentence. The following rule defining the predicate rental_order thus reads: "For all internal variables Cgroup, Ptime, Dtime, Pbranch, Dbranch, and Cust, there exists a rental order with the attribute values Cgroup, Ptime, Dtime, Pbranch, Dbranch, and Cust, if there is an activity of the type "Create rental reservation" whose input parameters have the values Cgroup, Ptime, Dtime, Pbranch, Dbranch, and Cust, and the customer identified by Cust is not blacklisted, and the car group identified by Cgroup has enough rental capacity between the pick-up time (Ptime) and drop-off time (Dtime) requested":

```
rental_order(Cgroup, Ptime, Dtime, Pbranch, Dbranch, Cust) <-
activity(create_rental_reservation(Cgroup, Ptime, Dtime, Pbranch, Dbranch, Cust)),
not is_blacklisted(customer(Cust)), has_capacity(car_group(Cgroup), Ptime, Dtime)
```

This rule includes the additional predicates is_blacklisted and has_capacity. The first of them is defined through facts and the second one by means of another Prolog rule which is not specified here.

In the extended AORML, the predicate activity(create_rental_reservation(Cgroup, Ptime, Dtime, Pbranch, Dbranch, Cust)) is represented as the activity starting event type START Create_rental_reservation(Cgroup: String, Ptime: Date, Dtime: Date, Pbranch: String, Dbranch: String, Cust: String). The occurrence of the event of this type evaluates the rule's internal variables Cgroup, Ptime, Dtime, Pbranch, Dbranch, and Cust.

The only way how to model by means of OCL the retrieval from the agent's VKB of entity instances to be assigned to the corresponding internal variables, like the instances of Customer and CarGroup in reaction rule R2 of Figure 3-9, seems to be through named contextual instances (recall from section 3.5 that we allow an OCL expression to have several named contextual instances). For example, provided that reaction rule R2 is specified in the context of the named contextual instances customer and carGroup of the respective types, the precondition of the rule, corresponding to the conjunction of the predicates not is_blacklisted(customer(Cust)), has_capacity (car_group(Cgroup), Ptime, Dtime), looks like as follows:

```
Customer.allInstances->select(personID = Cust and not isBlacklisted)->includes(customer) and
CarGroup.allInstances->select(carGroupID = Cgroup and hasCapacity(Ptime, Dtime))->includes(carGroup)
```

[7] An agent's *virtual knowledge base* (*VKB*) is called "virtual" because it is not necessarily implemented as a classical knowledge base. It can be implemented either as some relational, object-relational, or object-oriented database, ERP- or EAI-system, or object-oriented framework such as COM™ or CORBA™.

Since there is just one combination of (contextual) instances of Customer and CarGroup for which the precondition expression of reaction rule R2 evaluates to true *if* the customer is not blacklisted and there is enough rental capacity in the requested car group, the internal variables of this reaction rule are evaluated and the rule's mental effect and action parts are performed only once.

According to [Sterling86], a *relation scheme* specifies the role that each position in the relation is intended to represent. For example, the relation scheme car_group_of_rental_order(RentalOrder, CarGroup) corresponds to the association between the object types RentalOrder and CarGroup. The following Prolog rule specifies the creation of the association between an instance of RentalOrder and the instance of CarGroup that is identified by the value of its identifier attribute Cgroup:

```
car_group_of_rental_order(RentalOrder, CarGroup) <-
rental_order(Cgroup, Ptime, Dtime, Pbranch, Dbranch, Cust), car_group(Cgroup).
```

This rule can be read as follows "Whenever there is a rental order with the attribute values Cgroup, Ptime, Dtime, Pbranch, Dbranch, and Cust, where Cgroup identifies an instance of CarGroup, there is an association between the rental order and car group". The schema of reaction rule R2, which we are describing, thus also includes the internal variable corresponding to the instance of RentalOrder created by the rule that is retrieved from the agent's VKB. Analogously, a rule specifying the creation of the association between an instance of RentalOrder and the corresponding instance of Customer could be defined.

The mechanism of evaluating internal variables that we have been describing enables straightforward definition of loops. For example, the following Prolog clause for reaction rule R2 in Figure 3-9 defines a loop where the predicate allocate_a_car, standing for the activity starting action of the type START ACTIVITY Allocate_a_car(rentalOrder), is evaluated for each instance of RentalOrder having the status isReserved:

```
allocate_a_car(rentalOrder(RentalOrderID)) <-
isReserved(rentalOrder(RentalOrderID))
```

In OCL, the precondition of the same loop can be defined by the following logical expression that is evaluated in the context of an instance of RentalOrder where the stereotype name rentalOrder stands for a contextual instance:

```
RentalOrder.allInstances->select(isReserved)->includes(rentalOrder)
```

As the corresponding Prolog predicate presented above, this precondition is evaluated for all contextual instances of RentalOrder retrieved from the agent's VKB that have the status isReserved.

3.6.4. Visualization of Preconditions

As we will see in section 3.8.5.2, the precondition defined for an activity type is visualized as the precondition of the reaction rule preceding the activity type. The precondition of a reaction rule can be visualized by one or more ordinary incoming arrows from status and/or intensional predicate(s), which are attached to the corresponding informational entity type(s), to the reaction rule symbol. If there are two or more incoming arrows from predicates, the target reaction rule implicitly represents a logical conjunction for the predicates. A little cross at the beginning of an incoming arrow stands for negation. For example, in Figure 3-9 the precondition of reaction rule R2 within the activity type "Create rental reservation" specifies the conjunction of (1) the negation of the status predicate isBlacklisted applied to the representation of an agent of the type Customer; (2) the intensional predicate hasCapacity applied to an object of the type CarGroup. A precondition arrow may be augmented by a relevant OCL expression as is demonstrated in Figure 3-9. *The OCL expression attached to the precondition arrow originating in a status predicate constitutes the equation part of the* select-*operation* described in section 3.6.3. For example, the equation part pertaining to the select-operation Customer.allInstances->select(personID = Cust and not isBlacklisted)->includes(customer) consists of just one equation personID = Cust. *The OCL expression attached to the precondition arrow originating in an intensional predicate constitutes the logical expression part of the* select-*operation* described in section 3.6.3, like the logical expression carGroupID = Cgroup and hasCapacity(Ptime, Dtime) pertaining to the select-operation CarGroup.allInstances->select(carGroupID = Cgroup and hasCapacity(Ptime, Dtime))->includes(carGroup). A precondition is defined in terms of internal variables of the reaction rule. However, as a simplification *we allow to represent a precondition in terms of input variables defined for the enclosing activity types*.

An incoming arrow from a predicate may also be connected to the diamond symbol with or without the symbol 'X' inside respectively standing for an exclusive and inclusive disjunction of predicates. An example is the inclusive disjunction of the status predicates hasAdSpace and hasAlternativeAdSpace that is checked by reaction rule R34 in Appendix G.

If the precondition arrow originates in the symbol for an informational entity type instead of a predicate, the only intention of the precondition is to retrieve from the agent's VKB one or more instances of the informational entity type as is determined by the OCL expression attached to the precondition arrow.

3.6.5. Specification and Visualization of Mental Effects

Mental effects of a reaction rule are defined in terms of the rule's internal variables. According to Table 3-3, different ***categories*** of mental effects can be distinguished based on their types CREATE, DELETE, and UPDATE. The source data items of a mental effect are specified in the second column of Table 3-3. A ***source data item*** is normally an internal variable of a reaction rule. However, in order to keep activity diagrams simple, *we also allow to employ input parameters defined for the enclosing activity types as source data items of a mental effect*. The third and fourth columns of Table 3-3 respectively specify the status that the entity instance to be created or updated must have and the type of the mental effect. The fifth column defines the mental effect as an OCL expression in terms of the source data items and status specification, if it exists.

Mental effects can be visualized by mental effect arrows. A ***mental effect arrow*** is an arrow with empty arrowhead which specifies a change in the agent's beliefs and/or commitments/claims. Types of mental effects – CREATE, DELETE, and UPDATE – are distinguished by augmenting a mental effect arrow with the letter 'C', 'D', or 'U', respectively, which may be followed by the effect's number of order in parenthesis, as is shown in Figure 3-9. The implicit mental effect type is CREATE. The meanings of mental effect arrows are defined in Table 3-3 where typeSpecifier, possibly followed by its number of order, specifies the type of an informational entity affected by the mental effect and isStatus1 specifies the status that the instance to be created or updated must have. The definition of a mental effect may include a user-defined logical OCL expression which is distinguished using *italic* in Table 3-3. A user-defined expression should be represented explicitly as an augmentation of a mental effect arrow.

There are two categories of mental effects of the type "Create an entity" defined in Table 3-3. A mental effect of the first category specifies the creation of an instance of an informational entity type so that the explicitly specified logical expression included by the mental effect definition is true. For example, the following definition of a mental effect of the type "Create an entity" of reaction rule R2 in Figure 3-9 includes the user-defined OCL expression, shown in *italic*, which ensures that the attributes of the instance created equal to the values of the respective source data items:

RentalOrder.allInstances->exists(r: RentalOrder | *r.carGroupID = cgroup and r.pickUpTime = ptime and r.dropOffTime = dtime and r.pickUpBranchID = pbranch and r.dropOffBranchID = dbranch* and r.isAllocated)

Specification of a status change is not included by a user-defined logical expression attached to the mental effect arrow because the status is determined by the destination of the mental effect arrow. A mental effect of the second category differs from the first one in that it implicitly specifies copying the attribute values of the entity to be created from the respective attributes of the source data item of the same type.

A mental effect of the category "Update an entity" specifies analogously the update of an entity instance so that the attributes of the instance equal to the respective attributes of the source data item of the same type, the explicitly specified logical expression included by the mental effect, if any, is true, and the entity instance created has the status specified, if there is any. According to Table 3-3, if a mental effect of the category "Update an entity" specifies only a status change, the mental effect expression may be shortened like r.isEffective, where the input parameter r refers to the corresponding instance of RentalOrder. In such a case, the augmentation of the mental effect arrow is not needed.

In Table 3-3, two categories of mental effects of deleting an entity are defined. A mental effect of the first category specifies the deletion of the entity that is provided as a source data item, while within a mental effect of the second category the instance to be deleted is identified by an explicit user-defined logical expression.

A mental effect of creating a relationship specified in Table 3-3 employs as source data items the instances of the entity types the relationship is to be formed between. For example, the mental effect arrows of reaction rule R2 in Figure 3-9 specify the creation of the relationships between the instance

of RentalOrder created by the rule and the corresponding instances of CarGroup and Customer. The internal variables of reaction rule R2 serve as source data items of these mental effects. Deletion of a relationship can be specified by simply negating the expression for its creation and/or augmenting the corresponding mental effect arrow with the letter 'D'.

As is shown in Table 3-3, mental effects of the categories "Create a *to-do*-commitment/claim" and "Create a *stit*-commitment/claim" are instances of the corresponding commitment/claim types that inherit their attributes from the abstract object classes *ToDoCommitmentClaimType* and *STITCommitmentClaimType*, respectively, which will be explained in section 3.8.3.3. Since a *stit*-commitment/claim type is viewed as an anonymous class [OMG03b], it can be referred to by only navigating to it from the contextual agent instance, identified by self, in the direction of the rolename stitCommitmentClaim, as is shown in the last row of Table 3-3 and in Figure 3-15. Deletion of a commitment/claim is specified by negating the expression for its creation and/or augmenting the corresponding mental effect arrow with the letter 'D'.

It is important to notice here that if there is an ambiguity among internal variables of a reaction rule serving as source data items of the rule's mental effect, i.e. if there are two or more internal variables of the same type, *the mental effect must be defined explicitly* in terms of the rule's internal variables.

Table 3-3. Definitions of mental effect categories. Square brackets […] stand for optionality.

Description of the category	Source data items	Status	Mental effect type	Definition
Create an entity	variable1 : typeSpecifier1, ... variableN : typeSpecifierN	[isStatus1]	CREATE	typeSpecifier.allInstances->exists (v : typeSpecifier | *logical-expression-with*-v [and v.isStatus1])
Create an entity	entity : typeSpecifier	[isStatus1]	CREATE	typeSpecifier.allInstances->exists (v : typeSpecifier | v.attr1 = entity.attr1 and ... and v.attrN = entity.attrN and *logical-expression-with*-v [and v.isStatus1])
Update an entity	entity : typeSpecifier	[isStatus1]	UPDATE	typeSpecifier.allInstances->exists (v : typeSpecifier | v = entity and v.attr1 = entity.attr1 and ... and v.attrN = entity.attrN and *logical-expression-with*-v [and v.isStatus1]) OR entity.isStatus1
Delete an entity	entity : typeSpecifier	-	DELETE	not (typeSpecifier.allInstances->exists (v : typeSpecifier | v = entity))
Delete an entity	variable1 : typeSpecifier1, ... variableN : typeSpecifierN	-	DELETE	not (typeSpecifier.allInstances->exists (v : typeSpecifier | *logical-expression-with*-v))
Create a one-to-one relationship	entity1 : typeSpecifier1, entity2 : typeSpecifier2	-	CREATE	entity1.typeSpecifier2 = entity2 and entity2.typeSpecifier1 = entity1 and *logical-expression*
Create a one-to-many relationship	entity1 : typeSpecifier1, entity2 : typeSpecifier2	-	CREATE	entity1.typeSpecifier2->includes(entity2) and entity2.typeSpecifier1 = entity1 and *logical-expression*
Create a many-to-one relationship	entity1 : typeSpecifier1, entity2 : typeSpecifier2	-	CREATE	entity1.typeSpecifier2 = entity2 and entity2.typeSpecifier1->includes(entity1) and *logical-expression*
Create a many-to-many relationship	entity1 : typeSpecifier1, entity2 : typeSpecifier2	-	CREATE	entity1.typeSpecifier2->includes(entity2) and entity2.typeSpecifier1->includes(entity1) and *logical-expression*
Create a *to-do*-commitment/claim	targetObject : OclAny, dueDate : Date, agentID1 : String, agentID2 : String	-	CREATE	todoCommitmentClaimTypeSpecifier. allInstances->exists (c : todoCommitmentClaimTypeSpecifier | *c.about = targetObject and c.dueBy = dueDate and sourceID = agentID1 and targetID = agentID2*)
Create a *stit*-commitment/claim	expression : OclExpression, dueDate : Date, agentID1 : String, agentID2 : String	-	CREATE	self.stitCommitmentClaim->exists (*achieve = expression and dueBy = dueDate and sourceID = agentID1 and targetID = agentID2*)

3.6.6. Operational Semantics of Activity Diagrams

An activity diagram of the extended AORML can be considered as a specification of a high-level state transition system where the state of an agent consists of two parts: its ***mental state*** (beliefs, memory of events, actions, and commitments/claims), and its ***activity state***. Modelling by activity diagrams of the extended AORML is thus based on the semantic framework of *Knowledge-Perception-Memory-Commitment* (KPMC) agents that we will extend with the operational semantics for activities.

The concept of KPMC agents is an extension of the knowledge- and perception-based (KP) agent model proposed in [Wagner96] and [Wagner98] and refined in [Wagner00b]. According to [Wagner98], the logic underlying the operational semantics of KPMC agents is the *logic of state transition systems*.

In the sequel, where L is a language (a set of formulas), then L^0 denotes its restriction to closed formulas (sentences). Elements of L^0_{Query}, i.e. closed query formulas, are also called ***if-queries***. A query that includes free variables is called an ***open query***.

According to [Wagner96] and [Wagner00b], in the core of a KPMC agent is an abstract ***knowledge system*** which consists of four languages and three operations. The languages are a knowledge representation language L_{KB}, a query language L_{Query}, an input language L_{Input}, such that $L_{\text{Input}} \subseteq L_{\text{Query}}$, and an answer language L_{Ans}. The operations are an inference relation $\vdash \in L_{\text{KB}} \times L^0_{\text{Query}}$, such that $X \vdash F$ holds if $F \in L^0_{\text{Query}}$ can be inferred from $X \in L_{\text{KB}}$, an update operation $\mathsf{Upd} : L_{\text{KB}} \times L^0_{\text{Input}} \rightarrow L_{\text{KB}}$, such that the result of updating $X \in L_{\text{KB}}$ with $F \in L^0_{\text{Input}}$ is the knowledge base $\mathsf{Upd}(X, F)$, and an answer operation $\mathsf{Ans} : L_{\text{KB}} \times L_{\text{Query}} \rightarrow L_{\text{Ans}}$, such that $X \vdash F$ *iff* $\mathsf{Ans}(X, F) = \mathsf{yes}$[8]. An answer $(c_1, ..., c_k)$ to an open query $F = F[x_1, ..., x_k]$ with free variables $x_1, ..., x_k$ can be viewed as an answer substitution $\sigma = \{x_1/c_1, ..., x_k/c_k\}$ so that the resulting instantiation $F\sigma = F[c_1, ..., c_k]$ holds in the current knowledge base state.

The *schema* of a KPMC agent is composed of a *knowledge system* described above, a communication event language or an *agent communication language* (ACL) L_{CEvt}, a perception or *environment event language* L_{PEvt}, which form together the event language $L_{\text{Evt}} = L_{\text{CEvt}} \cup L_{\text{PEvt}}$, and an *action language* L_{Act}. A ***KPMC agent*** consists of five components: a ***virtual knowledge base*** $VKB \in L_{\text{KB}}$, an *event queue* EQ being a list of instantiated event expressions $\varepsilon(U) \in L^0_{\text{Evt}}$, where U is suitable list of parameters, a *memory base* $MB \in L_{\text{Evt}} \cup L_{\text{Act}}$ (recording past events and actions), a *commitment/claim base* $CB \in L_{\text{KB}}$ (recording commitments/claims), and a set of *reaction rules* $RR \subseteq (L_{\text{CEvt}} \cup L_{\text{Act}}) \times L_{\text{Input}} \times L_{\text{Evt}} \times L_{\text{Query}}$ (encoding the behaviour of the agent).

Reaction rules of a KPMC agent have the following general form:

(**do**(α) | **sendMsg**[$\eta(V)$, i] | *Eff*)* ← (**recvMsg**[$\varepsilon(U)$, j])+ *Cond*

In the notation used for expressing the above form, "|" stands for a choice and "?" stands for optional, "*" means zero or more times, and "+" means one or more times. The event condition **recvMsg**[$\varepsilon(U)$, j] is a test whether the event queue EQ of the agent contains a perception or communication event of the form $\varepsilon(U) \in L^0_{\text{Evt}}$ created by some perception subsystem of the agent or sent by another agent j; the precondition $Cond \in L_{\text{Query}}$ refers to the agent's current knowledge state represented in its VKB and CB; $Eff \in L_{\text{Input}}$ is an epistemic (mental) effect formula specifying corresponding updates of the agent's VKB, CB, and MB; **sendMsg**[$\eta(V)$, i] sends the message $\eta(V) \in L_{\text{CEvt}}$ with parameters V to the receiver i; and **do**($\alpha(V)$), where $\alpha(V) \in L_{\text{Act}}$, calls a procedure realizing the action α with parameters V. KPMC agents can thus perform *epistemic*, *communicative*, and *physical actions*. We do not repeat here the operational semantics of reaction rules based on [Manna92] which is presented in [Wagner96].

We now define the ***agent state*** of a KPMC agent as $S = (X, EQ, A)$ where X stands for the agent's current knowledge state, EQ is the agent's event queue, and A represents the agent's activity state. The agent's knowledge state X is represented in its virtual knowledge base VKB, commitment/claim base CB, and memory base MB which were introduced above. According to [Wagner98], the agent's event queue EQ represents the agent's connection to the environment. It stores incoming messages from the agent's perception subsystems and from other agents. The stored events are consumed one by one triggering appropriate reactions. Finally, the agent's ***activity state*** A represents the execution of a set of

[8] Because of its built-in general Closed-World Assumption, e.g. a relational database answers an if-query by either yes or no.

atomic actions that has some duration. In order to represent an agent's activity state, we complement the schema of a KPMC agent by an activity specification language L_{Activity} and an activity query language $L^0_{\text{Activity, Query}}$. Since elements of the latter are closed query formulas, it is only possible to pose a query whether the agent is in a specific activity state. The agent's activity state A may include a number of ***subactivity states***. It can be represented as the recursive logical term $s(U, Y) \in L_{\text{Activity}}$ where s denotes an activity state, U is a data structure recording the names, types, and values of the activity's input parameters (it may be empty), and Y specifies a number of parallel subactivity states $s_1(U, Y)$, ..., $s_n(U, Y)$ where $n >= 0$ and $Y \in L_{\text{Activity}} \cup \varnothing$. The agent's outermost state s is termed its ***root activity state***. The agent can be in only one root activity state at a time. In other words, only one outermost activity can be under execution at any moment of time.

Transferring an agent from one activity state to another can be specified by means of a reaction rule. For this purpose, we are complementing the action part of a reaction rule of the form presented above by the action terms **start**(β(*T*)) and **end**(β), where $\beta(T) \in L_{\text{Activity}}$ specifies the activity β to be started with the input parameters *T*. The actions defined by these action terms make the agent respectively to enter and exit the activity state corresponding to the activity β. In addition to the action term **start**(β(*T*)), which transfers the agent to the root activity state *s* corresponding to the activity β so that $A = s(U, \varnothing)$, there is an action term $\textbf{start}_{\text{sub}}(\beta(T))$ which adds to the agent's current activity state $A = s(U, Y)$ the state *t* of the *parallel* subactivity specified by $\beta(T) \in L_{\text{Activity}}$ so that $Y = Y \cup t(V, \varnothing)$. Entering the activity state of β creates the *activity border event* **start-of**(β) which is added to the agent's event queue *EQ*. Analogously, when an agent exits the activity state corresponding to the activity β, the activity border event **end-of**(β) is generated and added to the agent's event queue *EQ*. The activity border events **start-of**(β) and **end-of**(β) are consumed from the event queue as any other events. There are also action terms **remove**(β) and **remove-all** which respectively remove the representation of the activity β and the representations of all activities from the logical term $s(U, Y)$. The latter case results in $A = \varnothing$. Reaction rules of a KPMC agent thus take the following form:

(**do**(α(*V*)) | **sendMsg**[η(*V*), *i*] | **start**(β(*T*)) | $\textbf{start}_{\text{sub}}(\beta(T))$ | **end**(β) | **remove**(β) | **remove-all** | *Eff*)*
← (**recvMsg**[ε(*U*), *j*])+ (**start-of**(β) | **end-of**(β))* *Cond*

As the formula above reflects, one or more components of a reaction rule, with the exception of the event condition and precondition (which may be empty, i.e. *Cond* = true), may be omitted from the rule. Differently from a reaction rule without activities, the precondition $Cond \in L_{\text{Query}} \cup L^0_{\text{Activity, Query}}$ in a reaction rule of the form presented above may include an *if-query on the agent's activity state*. A reaction rule may have multiple event conditions and/or action terms that are implicitly connected with logical conjunction(s). If the action part of a rule is empty, the rule simply does nothing.

It is assumed that any activity concludes with the **end**(β) term. *This enables two important features*. Firstly, it allows to create executable specifications that include unspecified "dummy" activity types, each of which consists of just one reaction rule of the type **end**(β) ← **start-of**(β). Secondly, since an activity can be viewed as a "black box" with a duration, it is possible to use for achieving the activity's goal basically any procedure or method provided that it concludes with the **end**(*a*) term.

Please notice that reaction rules specified in the above form are *atomic* as compared to reaction rules represented using the language defined in Appendix B which consists of one or more atomic reaction rules.

Semi-formally, the execution model of an extended KPMC agent consists of the following steps:

1. Get the next event from the event queue *EQ*, and check whether it triggers any reaction rules by matching it with the *unmatched* events in event conditions of all rules in *RR*. If the event matches with some event in the event condition of a rule, mark this event as a matched one. If *all events* of a reaction rule have been matched, mark the reaction rule as a triggered one. If the event condition cannot be matched with any reaction rule, repeat step 1.
2. For each of the triggered reaction rules, evaluate its epistemic condition *Cond* like a query. For each answer substitution $\sigma \in \text{Ans}(VKB \cup CB \cup A, Cond)$, form the corresponding action/effect tuple by instantiating all free variables in the outgoing message expression η(*V*) and/or in the action expression α(*V*) and/or in the activity starting expression β(*T*) and/or in the mental effect formula *Eff* accordingly (in the case of an epistemic reaction rule, the 'empty' action *noAct* is used to form the action/effect tuple [*noAct*, *Eff*σ]).
3. For each of the resulting action/effect tuples, perform the communicative action **sendMsg**[η(*V*σ), *i*] and/or the physical action **do**(α(*V*σ)) and/or assimilate the mental effect

Effσ into the agent's *VKB*, *CB*, and *MB* yielding Upd(*VKB* ∪ *CB* ∪ *MB*, *Effσ*). If the tuple contains the **start**(β(*T*σ)) expression, create the logical term *s*(*U*, ∅) corresponding to the activity β, copy the values of the input parameters in *T*σ to the corresponding elements of the logical term's data structure *U*, equalize the agent's activity state with the logical term so that *A* = *s*(*U*, ∅), and add the activity border event **start-of**(β) to the agent's event queue *EQ*. If the tuple contains the $\mathbf{start}_{sub}$(β(*T*σ)) term, perform the following substeps:

a) if *A* = ∅, create the logical term *s*(∅, ∅) corresponding to the implicit superactivity χ, equalize the agent's activity state with the logical term so that *A* = *s*(∅, ∅), and add the activity border event **start-of**(χ) to the agent's event queue;
b) create the logical term *t*(*V*, ∅) corresponding to the activity β, copy the values of the input parameters in *T*σ to the corresponding elements of the logical term's data structure *V*, and complement the logical term *A* = *s*(*U*, *Y*) so that *Y* = *Y* ∪ *t*(*V*, ∅);
c) add the activity border event **start-of**(β) to the agent's event queue *EQ*.

If the tuple contains the **end**(β) term, where β is any activity, perform the following substeps:

a) remove the representation of the (sub)activity β from the logical term *A* = *s*(*U*, *Y*) describing the agent's activity state;
b) add the activity border event **end-of**(β) to the agent's event queue *EQ*;
c) while there are activities where all parallel subactivities have ended, repeat recursively: if all parallel subactivities of an activity β have ended, remove the representation of the activity β from the logical term *A* = *s*(*U*, *X*) and add the activity border event **end-of**(β) to the agent's event queue *EQ*.

4. Return to step 1.

According to [Wagner00b], the temporal behaviour of an agent can be described by means of transitions between agent states. We now modify the transition system semantics of a reactive agent presented in [Wagner96] and [Wagner00b] by giving the following definition:

Let *S* = (*X*, *EQ*, *A*) be an agent state, ε an event, and RR_ε a function that updates a knowledge state *X* with all mental effects of reaction rules in *RR* which are triggered by ε and whose condition holds in *X* and *A*, and also updates an activity state *A* as specified by reaction rules in *RR* which are triggered by ε and whose condition holds in *X* and *A*. If the only applicable reaction rule of the triggered rule set is *r*, we can use *r* instead of RR_ε. An empty event queue *EQ* or an empty activity state *A* is represented by [], an event queue with head *q* and tail *Q* by *q* : *Q*, and adding an element *q* to an event queue *Q* by *Q* + *q*. Then we have two kinds of transitions transforming an agent state *S* = (*X*, *EQ*, *A*):

1. Perception

$(X, EQ, A) \rightarrow^{\varepsilon} (X, EQ + \varepsilon, A)$

2. Reaction

$(X, \varepsilon : EQ, A) \rightarrow^{RR\varepsilon} (RR_\varepsilon(X), EQ, RR_\varepsilon(A))$

According to [Wagner96], we assume that all state transitions are atomic. That is, once an agent is getting involved in a transition, other agents can not influence the transition or observe intermediate points of it. In the execution model of an extended KPMC agent, the same also applies to activities: once an agent is getting involved in a transition within some activity, other activities of the agent can not influence the transition or observe intermediate points of it. This implies two things: a) an activity indeed corresponds to a transaction as we suggested in section 3.6.1 b) the simultaneous execution of actions can be serial in the following sense: if two elementary actions, say α_1 and α_2 are executed concurrently, then the net effect is either that of α_1 followed by α_2, or α_2 followed by α_1. This enables serial processing of events described by step 1 of the execution model of an extended KPMC agent.

An ***execution history*** of an agent is a chain of state transitions $S^0 \rightarrow^{\tau 0} S^1 \rightarrow^{\tau 1}$, … where each τ_i corresponds to either a perception or reaction transition described above. A history can be finite or infinite. By definition, a finite history ends in a state [Wagner96]. Examples of formal verification of state transition systems by means of assertional reasoning are provided in [Wagner96].

As an example, the reaction rules modelled in Figure 3-9 can be represented as the following set of atomic reaction rules *RR* = {r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}}:

r_1: **start**(*Manage_car_reservation*(...)) ← **recvMsg**[*request*(*provideCar*(...)), *customer*]

r_2: **start**$_{sub}$(*Check_the_customer_for_blacklistedness*(...)) ← **start-of**(*Manage_car_reservation*)

r_3: **end**(*Check_the_customer_for_blacklistedness*) ← **start-of** (*Check_the_customer_for_blacklistedness*)

r_4: **start**$_{sub}$(*Create_rental_reservation*(...)) ← **end-of**(*Check_the_customer_for_blacklistedness*)

r_5: *isReserved*(*RentalOrder*(...)), **sendMsg**[*agree*(*provideCar*(*RentalOrder*(...))), *customer*], **end**(*Create_rental_reservation*) ← **start-of**(*Create_rental_reservation*), not *isBlacklisted*(...), *hasCapacity*(...)

r_6: **sendMsg**[*refuse*(*provideCar*(...)), *customer*], **end**(*Create_rental_reservation*) ←

start-of(*Create_rental_reservation*), *isBlacklisted*(...)

r_7: **sendMsg**[*refuse*(*provideCar*(...)), *customer*], **end**(*Create_rental_reservation*) ←

start-of(*Create_rental_reservation*), not *hasCapacity*(...))

r_8: **start**$_{sub}$(*Allocate_cars*) ← **end-of**(*Create rental reservation*)

r_9: **start**$_{sub}$(*Allocate_a_car*(...)) ← **start-of**(*Allocate_cars*)

r_{10}: **end**(*Allocate_a_car*) ← **start-of**(*Allocate_a_car*)

We are now introducing an example of an execution history of an agent of the type Pick-UpBranch. In the interests of simplicity, in this example we omit the data structure *U* from the logical term *A* = *s*(*U*, *Y*) representing the activity state *A* of the agent which is initially empty. When the agent receives from the *customer* a request to provide him/her with a car (i.e., perceives a communicative action event of the type *request*(*provideCar*(...))), reaction rule r_1 is triggered. According to this reaction rule, an activity of the type *Manage_car_reservation* is started and the agent is switched to the corresponding root activity state. In reaction to "perceiving" the start of the activity of the type *Manage_car_reservation*, reaction rule r_2 starts its subactivity of the type *Check_the_customer_for_blacklistedness*, and the activity state of the agent is accordingly changed to *Manage_car_reservation*(*Check_the_customer_for_blacklistedness*). Reaction rule r_3 ends the "dummy" subactivity *Check_the_customer_for_blacklistedness* in reaction to its starting event. After the subactivity of the type *Check_the_customer_for_blacklistedness* has ended, reaction rule r_4 starts a subactivity of the type *Create_rental_reservation*, changing the agent's activity state into *Manage_car_reservation*(*Create_rental_reservation*). Reaction rule r_5 is triggered by the activity starting event **start-of**(*Create_rental_reservation*). Since the precondition of this reaction rule is true, a rental reservation having the status *isReserved* is created (for simplicity, creations of relationships have been omitted), an agreement to provide a car accompanied by the rental order is sent to the customer, and finally, the activity of the type *Create_rental_reservation* is ended. In the execution history below, all these actions are represented by the function $r_5(X)$. In response to the **end-of**(*Create_rental_reservation*) event, reaction rule r_8 starts a subactivity of the type *Allocate_cars*. The starting event of the latter, in turn, triggers reaction rule r_9 which starts a "dummy" activity of the type *Allocate_a_car*. The latter is ended by its only reaction rule r_{10}. The activities *Allocate_cars* and *Create_rental_reservation* are ended according to step 3 of the execution model of an extended KPMC agent when all their subactivities have ended. The events **end-of**(*Allocate_a_car*), **end-of**(*Allocate_cars*), and **end-of**(*Manage_car_reservation*) do not thus trigger any actions. The described execution history of an agent operating on the basis of the rule set *RR* defined above thus looks like as follows:

$(X, EQ, []) \rightarrow {}^{\textbf{recvMsg}[request(provideCar(...)),\ customer]} \rightarrow (X, EQ + \textbf{recvMsg}[request(provideCar(...)), customer], [])$

$(X, \textbf{recvMsg}[request(provideCar(...)), customer] : EQ, []) \rightarrow {}^{r1} \rightarrow (X, EQ, Manage\_car\_reservation)$

$(X, EQ, Manage\_car\_reservation) \rightarrow {}^{\textbf{start-of}(Manage\_car\_reservation)} \rightarrow (X, EQ + \textbf{start-of}(Manage\_car\_reservation), Manage\_car\_reservation)$

$(X, \textbf{start-of}(Manage\_car\_reservation) : EQ, Manage\_car\_reservation) \rightarrow {}^{r2} \rightarrow (X, EQ, Manage\_car\_reservation(Check\_the\_customer\_for\_blacklistedness))$

(X, EQ, *Manage_car_reservation*(*Check_the_customer_for_blacklistedness*)) → $^{\textbf{start-of}(\textit{Check\_the\_customer\_for\_blacklistedness})}$ → (X, EQ + **start-of**(*Check_the_customer_for_blacklistedness*), *Manage_car_reservation*(*Check_the_customer_for_blacklistedness*))

(X, **start-of**(*Check_the_customer_for_blacklistedness*) : EQ, *Manage_car_reservation*(*Check_the_customer_for_blacklistedness*)) → ^{r3} → (X, EQ, *Manage_car_reservation*)

(X, EQ, *Manage_car_reservation*) → $^{\textbf{end-of}(\textit{Check\_the\_customer\_for\_blacklistedness})}$ → (X, EQ + **end-of**(*Check_the_customer_for_blacklistedness*), *Manage_car_reservation*)

(X, **end-of**(*Check_the_customer_for_blacklistedness*) : EQ, *Manage_car_reservation*) → ^{r4} → (X, EQ, *Manage_car_reservation*(*Create_rental_reservation*))

(X, EQ, *Manage_car_reservation*(*Create_rental_reservation*)) → $^{\textbf{start-of}(\textit{Create\_rental\_reservation})}$ → (X, EQ + **start-of**(*Create_rental_reservation*), *Manage_car_reservation*(*Create_rental_reservation*)))

(X, **start-of**(*Create_rental_reservation*) : EQ, *Manage_car_reservation*(*Create_rental_reservation*)) → ^{r5} → ($r_5(X)$, EQ, *Manage_car_reservation*)

(X, EQ, *Manage_car_reservation*) → $^{\textbf{end-of}(\textit{Create\_rental\_reservation})}$ → (X, EQ + **end-of** (*Create_rental_reservation*), *Manage_car_reservation*)

(X, **end-of**(*Create_rental_reservation*) : EQ, *Manage_car_reservation*) → ^{r8} → (X, EQ, *Manage_car_reservation*(*Allocate_cars*))

(X, EQ, *Manage_car_reservation*(*Allocate_cars*)) → $^{\textbf{start-of}(\textit{Allocate\_cars})}$ → (X, EQ + **start-of** (*Allocate_cars*), *Manage_car_reservation*(*Allocate_cars*))

(X, **start-of**(*Allocate_cars*) : EQ, *Manage_car_reservation*(*Allocate_cars*)) → ^{r9} → (X, EQ, *Manage_car_reservation*(*Allocate_cars*(*Allocate_a_car*)))

(X, EQ, *Manage_car_reservation*(*Allocate_cars*(*Allocate_a_car*))) → $^{\textbf{start-of}(\textit{Allocate\_\_a\_car})}$ → (X, EQ + **start-of**(*Allocate_a_car*), *Manage_car_reservation*(*Allocate_cars*(*Allocate_a_car*)))

(X, **start-of**(*Allocate_a_car*) : EQ, *Manage_car_reservation*(*Allocate_cars*(*Allocate_a_car*))) → ^{r10} → (X, EQ, [])

(X, EQ, []) → $^{\textbf{end-of}(\textit{Allocate\_a\_car})}$ → (X, EQ + **end-of** (*Allocate_a_car*), [])

(X, **end-of**(*Allocate_a_car*) : EQ, []) → (X, EQ, [])

(X, EQ, []) → $^{\textbf{end-of}(\textit{Allocate\_cars})}$ → (X, EQ + **end-of** (*Allocate_cars*), [])

(X, **end-of**(*Allocate_cars*) : EQ, []) → (X, EQ, [])

(X, EQ, []) → $^{\textbf{end-of}(\textit{Manage\_car\_reservation})}$ → (X, EQ + **end-of** (*Manage_car_reservation*), [])

(X, **end-of**(*Manage_car_reservation*) : EQ, []) → (X, EQ, [])

The logical term $s(U, Y)$ for representing an activity state A can be implemented by using e.g. dynamic lists consisting of activity descriptors according to the principles of creating dynamic lists we have presented in [Tamm96]

3.6.7. Activity Modelling Language

As we saw in sections 3.6.1 – 3.6.6, a ***business process type*** can be defined from the perspective of a focus agent as a sequence of one or more reaction rules where each reaction rule is formulated in terms of its internal variables. This means that a business process type is ***closed*** in the sense that *to each action event or non-action event perceived by an agent within a business process of the given type correspond one or more actions performed by the agent within the same business process*.

A sequence of reaction rules defining a business process type can be equivalently specified using a semi-formal ***activity modelling language*** that enables to determine the structure and invocation order of activities. This ensures an important feature of activity diagrams ─ their ***executability***. The activity modelling language specifies a reaction rule as consisting of the following components:

- one or more *triggering events*;

- an optional *precondition*, consisting of one or more conditions;
- one or more *actions*, **and/or**
- a *postcondition*, consisting of one or more mental effects.

The *event part* defines the template for the types of action and non-action events to be processed by the rule. Please notice that the activity modelling language defined in Appendix B also allows for inclusive and exclusive logical disjunctions of event expressions. This is possible because reaction rules in the activity modelling language are not *atomic* like were reaction rules defined and used in section 3.6.6. This is to say, to a reaction rule in the activity modelling language correspond two or more atomic reaction rules. For example, to a reaction rule in the example presented at the end of this subsection correspond the atomic reaction rules r_5, r_6, and r_7, which were specified in section 3.6.6.

The *precondition part* specifies by means of OCL the conditions under which the action(s) prescribed by the rule is (are) executed. The *action part* consists of one or more elements of the following types:

- communicative action;
- physical action;
- activity starting action (START ACTIVITY activityReference construct);
- CANCEL ACTIVITY or CANCEL PROCESS constructs (to be described in section 3.8.5.3).

In the activity modelling language, each reaction rule is defined in the context of the enclosing agent or agent type. This enables the rule to access informational entities and their attributes and relationships within the agent instance. Additionally, a reaction rule can be defined in the context of instances of one or more informational entity types that are accessed by the rule's precondition, like the named instances carGroup and customer in the example below.The OCL constructs *formalParameterList* and *actualParameterList* of the activity modelling language are specified in terms of internal variables of the reaction rule.In the activity modelling language, the internal variables of a reaction rule are defined as formal parameters of the rule. When the reaction rule starts an activity, the values of the rule's internal variables are passed as actual parameters to the input parameters of the activity. For example, when an activity of the type "Manage car reservation" is started in a business process of the type depicted in Figure 3-9, the values of the internal variables of rule R1 are passed to the corresponding input parameters of the activity as is shown in the figure.

The activity modelling language is represented in Appendix B in the form of an EBNF grammar where "|" stands for a choice and "?" stands for an option, and "*" and "+" denote the repetition of a construct of zero or more and one or more times, respectively. The grammar references the grammar for the extended subset of OCL that is presented in Appendix A. The constructs of the extended OCL subset grammar are distinguished by representing them using *italics*.

The expressive power of the activity modelling language is the same as that of activity diagrams. This means that any activity diagram can be represented by means of the activity modelling language and vice versa. For example, the activity type "Manage car reservation" in Figure 3-9 can be represented in the activity modelling language as the following sequence of reaction rules:

```
CONTEXT Pick-Up Branch
ON RECEIVE MESSAGE request(provideCar(Cgroup: String, Ptime: Date, Dtime: Date, Dbranch: String,
SenderID: String))
START ACTIVITY Create_rental_reservation(Cgroup, Ptime, Dtime, self.agentID, Dbranch, SenderID)

CONTEXT Pick-Up Branch
ON START ACTIVITY Create_rental_reservation(Cgroup: String, Ptime: Date, Dtime: Date, Pbranch: String,
Dbranch: String, Cust: String)
START ACTIVITY Check_the_customer_for_blacklistedness(Cust)

CONTEXT Pick-Up Branch
ON START ACTIVITY Check_the_customer_for_blacklistedness(Cust: String)
END ACTIVITY Check_the_customer_for_blacklistedness

CONTEXT Pick-Up Branch
ON END ACTIVITY Check_the_customer_for_blacklistedness
START ACTIVITY Create_rental_reservation(Cgroup, Ptime, Dtime, Pbranch, Dbranch, Cust)

CONTEXT Pick-Up Branch, carGroup : CarGroup, customer : Customer
def: rentalOrder: RentalOrder
ON START ACTIVITY Create_rental_reservation
(Cgroup: String, Ptime: Date, Dtime: Date, Pbranch: String, Dbranch: String, Cust: String)
IF Customer.allInstances->select(personID = Cust and not isBlacklisted)->includes(carGroup) and
CarGroup.allInstances->select(carGroupID = Cgroup and hasCapacity(Ptime, Dtime))->includes(customer)
```

```
THEN EFFECT RentalOrder.allInstances->exists(ro: RentalOrder | ro.carGroupID = Cgroup and
ro.pickUpTime = Ptime and ro.dropOffTime = Dtime and ro.dropOffBranchID = Dbranch and
ro.pickUpBranchID = Pbranch and rentalOrder = ro) and
rentalOrder.carGroup->exists(cg : CarGroup | cg = carGroup and cg->includes(rentalOrder)) and
rentalOrder.customer->exists(c : Customer | c = customer and c->includes(rentalOrder))
SEND MESSAGE agree provideCar(rentalOrder) TO Cust
END Create_rental_reservation
ELSE
SEND MESSAGE refuse provideCar(Cgroup, Ptime, Dtime, Dbranch) TO Cust
END Create_rental_reservation

CONTEXT Pick-Up Branch
ON END ACTIVITY Create_rental_reservation
START ACTIVITY Allocate_cars

CONTEXT Pick-Up Branch
ON START ACTIVITY Allocate_cars
START ACTIVITY Allocate_a_car

CONTEXT Pick-Up Branch
ON START ACTIVITY Allocate_a_car
END ACTIVITY Allocate_a_car
```

3.7. ANALYSIS UTILIZING USE CASES WITH GOALS

The methodology of agent-oriented modelling proposed by us consists of the steps of *analysis* of the selected problem domain and *design* of a socio-technical system for it. Different modelling techniques, like e.g. ***i**** [Yu95a] or Resource-Event-Agent (REA) modelling framework of [McCarthy82], can be used for the step of analysis. In [Taveter02a], we have studied the applicability of ***i****, which is reviewed in section 2.1.4, for the domain analysis step. However, in this thesis we make use of *goal-based use cases* proposed in [Cockburn97a] and [Cockburn97b] for analyzing the problem domain at hand because their transformation into the corresponding models of design, represented in the extended AORML, is very straightforward. Moreover, in [Gottesdiener99] it is claimed that capturing of business rules should be done concurrently with the development of use cases because "behind every use case are business rules at work". In section 3.7.1, we describe how goal-based use cases can be used for analyzing a problem domain. In section 3.7.2, we illustrate the applying of goal-based use cases to the step analysis by using examples from the domain of the EU-Rent car rental company which was described in section 3.1.

3.7.1. Adaptation of Goal-Based Use Cases to Agent-Oriented Modelling

Use cases as such were originally introduced by Jacobson in [Jacobson92]. In [Cockburn97a] and [Cockburn97b], Cockburn proposes an extended version of use cases which he calls "use cases with goals". In [Cockburn97a], he defines a use case as "*a collection of possible sequences of interactions between the system under discussion and its external actors, related to a particular goal*". Goal-based use cases are elaborated on in [Cockburn01].

We employ goal-based use cases for two purposes:

- to model business process types for which a socio-technical system defined in section 1.7 is to be designed by using the extended AORML;
- to document the design models expressed in the extended AORML.

The stated purposes imply that business process modelling by goal-based use cases is an *iterative process* in the sense that in the course of the design phase we often have to return from the models of design to the corresponding models of analysis and revise them.

A goal-based use case consists of a ***primary actor***, the ***system under discussion***, and optionally one or more ***secondary actors***. According to [Cockburn01], when use cases document an organization's business processes, the *system under discussion* is the organization itself or one of its subsystems, like a department. The *external primary actors* are the actors whose goals the organization is to satisfy. They include the company's customers and perhaps their suppliers. The external primary actors form a part of the company's *stakeholders* which are the company shareholders, customers, vendors, and government regulatory agencies. A *secondary* or a *supporting actor* is an external actor that provides a service to the system under design. In parallel with the identification of primary actors, the ***triggering events*** that the organization must respond to should be identified [Cockburn01].

The business use cases described within our approach are the so-called ***white-box use cases*** [Cockburn01] because we look inside the organization whose business processes are to be modelled and discuss the behaviours of its internal actors. In modelling the behaviour of an internal actor initiated by another internal actor, we view the latter as a primary actor, i.e. *we change the focus from one internal actor to another*. This means that an internal actor may also serve as a primary or secondary actor. In this sense our business use cases differ from the business use cases described in [Cockburn01] where both primary and secondary actors in business use cases can be only external actors.

Cockburn notices in [Cockburn97a]: "It turns out that the system is itself an actor, and so the communication model needs only work with actors". Since 'actor' and 'agent' are synonyms for our purposes, the communication model of goal-based use cases should work in agent-oriented modelling, as well, "even though broadcast and asynchronous communications are omitted" in goal-based use cases [Cockburn97a]. It is, however, possible to model broadcast communication by using ***actor roles*** [Cockburn01] like the 'pick-up branch' or 'branch-proposer' in the example of car rental. Actor roles correspond to *agent roles* in the metamodel presented in section 3.3. We will provide a full example that includes actor roles in section 3.7.2. It is also possible to model asynchronous communication by goal-based use cases by assuming the presence of an *event queue* within the actor in focus.

According to [Cockburn97a], each actor has a set of ***responsibilities***. To carry out those responsibilities, it sets some goals. To reach a goal, it performs some actions. According to [Cockburn01], there is a need to describe three sorts of actions:

- an *interaction* between two agents;
- a *validation* (to protect a stakeholder);
- an *internal state change* (on behalf of a stakeholder).

Actions understood this way can be mapped to the agent-oriented notions of AOR modelling. In AORML [Wagner03a], an interaction is understood as a sequence of actions performed by the focus agent that are perceived as events by other agent(s) and the other way round, a validation is checking of some condition in the agent's VKB, and an internal state change corresponds to a mental effect.

It is further explained in [Cockburn97a] and [Cockburn01] that an actor can be a person, an organization, or a machine. The internal actor can be the system under discussion, a subsystem, or an object. The system under discussion consists of subsystems, which consist of objects. Actors have behaviour(s). The top-level behaviour is responsibility. It contains goals, which contain actions. An ***interaction*** between two actors is a kind of action that connects the actions of one actor with another. In other words, an interaction is one actor's goal calling upon another actor's (or its own) responsibility. If the second actor does not deliver its responsibility, for whatever reason, the primary actor has to find another way to deliver its goal. This is termed a "backup action".

Internal and external actors straightforwardly correspond to internal and external agents in AOR modelling, as they were defined in section 3.4.1. A "person" is a human agent, an "organization" constitutes an institutional agent, and a "machine" is equivalent to an artificial agent. A "subsystem" of the institutional agent in focus, e.g. of an enterprise, can be interpreted as its internal human, institutional, or artificial agent. The only difference between use cases with goals and AOR modelling with regard to actors is that in AOR modelling an object is not viewed as an actor, but rather an agent's VKB may include one or more objects that are manipulated by it. In the extended AORML, an agent's responsibilities correspond to the types of activities that the agent is capable of performing in response to perceiving action events of the corresponding types.

A ***compound interaction*** in goal-based use cases is understood as a recursive *sequence of interactions* [Cockburn97a]. At the bottom level, it consists of messages. A sequence has no branching or alternatives. It is therefore used to describe the past or a definite future, with conditions stated. Such a sequence is known as a ***scenario*** which is defined in [Cockburn97a] as a sequence of interactions happening under certain conditions, to achieve the primary actor's goal, and having a particular result concerning that goal. The interactions start from the triggering event and continue until the goal is delivered or abandoned, and the system completes whatever responsibilities it has concerning the interaction. A scenario in which the primary actor's goal is delivered and all the stakeholders' interests are satisfied is called the ***main (success) scenario*** [Cockburn01]. Failure of a scenario's step is handled by another scenario, or an extension scenario. According to [Cockburn97b], each ***extension scenario*** starts by stating where it picks up in the main scenario and what conditions are different. It then contains some lines and reverts back to the main scenario, or runs to a possibly different completion on its own.

In the light of the definition of a scenario, ***use case*** is more precisely defined in [Cockburn97a] as a collection of possible scenarios between the system under discussion and external actors, characterized by the goal the primary actor has toward the system's declared responsibilities, showing how the primary actor's goal might be delivered or might fail.

The "system under discussion" in the previous definition may be substituted for the "agent in focus" in an external AOR model. The interaction model of goal-based use cases is, however, simpler than interaction models of agents, described e.g. in [ACL97]. According to [Cockburn97b], the relationship between primary actor and system under discussion is that of ***client and server***. This is in contradiction with ***peer-to-peer*** relationships between actors which we have in agent-oriented approaches. Consequently, new modelling techniques are required for agent-oriented analysis of problem domains. Until they exist, the "traditional" modelling notations and approaches should be adapted to agent-oriented modelling. In particular, use cases with goals can be applied to agent-oriented analysis of problem domains by viewing the primary actor (i.e., the *client*) as the ***initiator*** and the system under discussion (i.e., the *server*) as the ***responder*** in agent-to-agent interactions. According to [Cockburn97b], the ***scope*** of a use case shows what system is being considered the system under discussion. In our approach, the scope shows the ***actor in focus*** whose behaviour is being described because we consider all systems and subsystems as human, institutional, or artificial agents. Since in the *peer-to-peer* interaction model a responder may also act as an initiator, a business process is modelled with goal-based use cases by changing the focus from one actor to another, so that for each actor of the problem domain there are one or more use cases where the actor is in focus.

Goals of use cases are divided into user goals and summary goals. The ***user goal*** is the goal the primary actor has in trying to get work done. Such a goal is achieved by what might be called a ***primary task*** or an "elementary business process". When we represent a business process by a goal-based use case in the extended AORML, a user goal attached to the primary actor is ***internalized*** by the actor (agent) in focus. For example, the user goal "a customer expects to have a car reserved for him/her to be picked up at the pick-up branch at the pick-up time and dropped off at the drop-off branch at the drop-off time" of the primary actor of use case 1 in section 3.7.2 becomes the following goal of the car rental company's pick-up branch: "reserve a car for the customer to be picked up at the pick-up branch at the pick-up time and dropped off at the drop-off branch at the drop-off time."

It is claimed in [Cockburn97b] that a user goal often corresponds to completing a transaction to the database. This is an interesting observation because in section 3.6.1 we noticed that an agent's activity, corresponding to some responsibility which is associated with a user goal, also often corresponds to completing a transaction.

Collections of user goals are ***summary goals*** like, for example, "rent a car" which involves user goals of use cases 1 ("Have a car reserved") and 2 ("Pick up the car") presented in Tables 3-4 and 3-5, respectively.

According to [Cockburn01], a use case may include a ***sub use case*** which refers to a goal at a lower goal level (i.e., a subgoal). For example, use case 1 ("Have a car reserved") in Table 3-4 consists of sub use cases 3 ("Check the customer for blacklistedness") and 5 ("Allocate cars"), presented in Tables 3-6 and 3-8, respectively, which refer to the corresponding goals. A sub use case is a ***subfunction*** of the primary task that it is included by. A *subfunction* is a step in a scenario that is below the main level of interest of the primary actor. Therefore, its goal, which is a sub-goal of some user goal, is attached to the actor in focus instead of the primary actor. Consequently, in order to achieve a user goal, the actor in focus may set one or more subgoals which are modelled as *goals of subfunctions*. Subfunctions may recursively consist of other subfunctions. A special group of sub use cases are subfunctions that are triggered by internal actors, like use case 5 ("Allocate cars") in Table 3-8.

Based on [Cockburn97b], we suggest the following form for writing steps of goal-based use cases where the first two elements are optional:

[<*time or sequence factor*>]…[<*condition*>]…<*actor*>…<*action*>…

This corresponds well to the general format for reaction rules in the metamodel presented in section 3.3, so that <*time or sequence factor*>, <*condition*>, and <*action*> stand for the event, precondition, and action, respectively, and <*actor*> represents the agent, performing the action. An extension scenario starts with either a <*time or sequence factor*> that triggers the extension scenario or a <*condition*> that must be true when the extension scenario is started. The triggering of a subfunction by an internal actor is also considered to be a <*time or sequence factor*>.

According to [Cockburn01], if several steps of a use case are to be repeated, the repetition <*condition*> is written either before or after the repeating steps. The statement about repetition is not numbered. In order to make goal-based use cases compatible with the extended AOR modelling, we require to include the steps to be repeated in a separate subfunction. For example, step 1 of use case 5 ("Allocate cars"), presented in Table 3-8, refers to the subfunction to be repeated "Allocate a car", presented in Table 3-9.

3.7.2. Applying Goal-Based Use Cases to the Example of Car Rental

The use cases in Tables 3-4 – 3-17 describe the business process type of car rental with an advance reservation. In Table 3-4, use case 1 ("Have a car reserved") is presented. The goal of the use case, "to have a car reserved for the customer to be picked up at the pick-up branch at the pick-up time and dropped off at the drop-off branch at the drop-off time", is given in its context in an informal way. It is semi-formalized in section 3.8.4 at the phase of design. The use case is modelled from the perspective of a customer with the pick-up branch of the car rental company in focus (*scope*). This means that the goal of the use case is the so-called *user goal*, the goal of the actor (i.e., the customer) trying to get work (*primary task*) done. The use case is triggered by receiving from a customer a request to have a car reserved. The customer is therefore called the *primary actor* of the use case. The actor 'headquarters' and actor role 'branch-proposer' are termed *secondary actors* because they are the ones from which the actor in focus (i.e., the pick-up branch) needs assistance to satisfy the user goal internalized by it. Another primary task, i.e. a use case that is triggered by the primary actor, is use

case 2 (“Pick up the car”). Either use case mentioned includes the main success scenario for satisfying its goal and a number of extension scenarios. For example, use case 2 includes the main scenario for the case where the customer does not have another EU-Rent car on rental, is more than 25 years old, has a valid driver’s license, and is physically able to drive the car safely, and the extension scenario for the case when any of the criteria mentioned is not satisfied.

The primary task “Have a car reserved (use case 1) includes as *subfunctions* use cases 3 (“Check the customer for blacklistedness”) and 5 (“Allocate cars”). The subfunction “Allocate cars”, in turn, includes as a subfunction use case 6 (“Allocate a car”). The latter recursively includes five more subfunctions. The primary task “Pick up the car” (use case 2) includes as a subfunction use case 4 (“Check the customer for another car”). As we learned in section 3.7.1, the goal of a *subfunction* is attached to the actor in focus. For example, as Table 3-6 shows, the goal “expect to become to know whether the customer is blacklisted” of the subfunction “Check the customer for blacklistedness” is attached to the pick-up branch.

A special group of subfunctions is made up of those triggered by internal actors. In the business process type of car rental with an advance reservation, to this group belongs use case 5 (“Allocate cars”) which is triggered by the internal actor ‘timer’ of the pick-up branch.

In Tables 3-4 through 3-17, the <*time or sequence factor*> and <*condition*> components of use case steps are distinguished by representing them in *italic*.

Table 3-4. Extended use case for the business process “Have a car reserved”.

USE CASE 1	Have a car reserved	
Goal in Context	A customer expects to have a car reserved for him/her to be picked up at the pick-up branch at the pick-up time and dropped off at the drop-off branch at the drop-off time.	
Scope & Level	Pick-up branch, primary task.	
Preconditions		
Success End Condition	The pick-up branch has reserved a car for the customer.	
Primary Actor **Secondary Actors**	Customer. Headquarters, branch-proposer.	
Trigger	A request by a customer to have a car reserved, specifying the car group, pick-up time, drop-off time, and drop-off branch.	
DESCRIPTION	**Step**	**Action**
	1	The pick-up branch checks the customer for blacklistedness with the headquarters (**Use Case 3**).
	2	*The customer is not blacklisted and there is enough rental capacity in the pick-up branch on the pick-up day*: the pick-up branch creates the rental reservation, informs the customer about the agreement, and commits to reserve a car for the customer.
	3	*It is the end of the day*: the pick-up branch allocates cars to rental reservations (**Use Case 5**).
EXTENSIONS	**Step**	**Branching Action**
	2a	*The customer is blacklisted or there is not enough rental capacity in the pick-up branch on the pick-up day*: the pick-up branch refuses the rental reservation, informs the customer about the refusal, and the business process ends.

Table 3-5. Extended use case for the business process "Pick up the car".

USE CASE 2	Pick up the car	
Goal in Context	A customer expects to pick up the car reserved for him/her at the pick-up branch at the pick-up time.	
Scope & Level	Pick-up branch, primary task.	
Preconditions	The pick-up branch has reserved a car for the customer.	
Success End Condition	The customer has picked up the car.	
Primary Actor **Secondary Actors**	Customer. Headquarters.	
Trigger	A pick-up request by the customer.	
DESCRIPTION	**Step**	**Action**
	1	The pick-up branch checks the customer for another car with the headquarters (**Use Case 4**).
	2	*The customer does not have another EU-Rent car on rental, is more than 25 years old, has a valid driver's license, and is physically able to drive the car safely*: the pick-up branch agrees to provide the customer with the car.
	3	The pick-up branch releases the car to the customer.
	4	The pick-up branch informs the headquarters about the pick-up.
EXTENSIONS	**Step**	**Branching Action**
	2a	*The customer has another EU-Rent car on rental, or is 25 years old or less, or does not have a valid driver's license, or is physically not able to drive the car safely*: the pick-up branch refuses to provide the customer with the car and the business process ends.

Table 3-6. Extended use case for the business process "Check the customer for blacklistedness".

USE CASE 3	Check the customer for blacklistedness	
Goal in Context	The pick-up branch expects to become to know whether the customer is blacklisted.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has received from the customer a rental reservation request.	
Success End Condition	The pick-up branch knows whether the customer is blacklisted.	
Primary Actor **Secondary Actors**	Customer. Headquarters.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The pick-up branch sends to the headquarters a query to validate that the customer is not blacklisted.
	2	The pick-up branch receives from the headquarters a reply of validation that the customer is not blacklisted, and registers the reply.
EXTENSIONS	**Step**	**Branching Action**
	2a	The pick-up branch receives from the headquarters a reply of validation that the customer is blacklisted, and registers the reply.

Table 3-7. Extended use case for the business process "Check the customer for another car".

USE CASE 4	Check the customer for another car	
Goal in Context	The pick-up branch expects to become to know whether the customer has another EU-Rent car on rental.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has received from the customer a pick-up request according to the rental reservation created by the pick-up branch.	
Success End Condition	The pick-up branch knows whether the customer has another EU-Rent car on rental.	
Primary Actor **Secondary Actors**	Customer. Headquarters.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The pick-up branch sends to the headquarters a query to validate that the customer does not have another EU-Rent car on rental.
	2	The pick-up branch receives from the headquarters a reply of validation that the customer does not have another EU-Rent car on rental, and registers the reply.
EXTENSIONS	**Step**	**Branching Action**
	2a	The pick-up branch receives from the headquarters a reply of validation that the customer has another EU-Rent car on rental, and registers the reply.

Table 3-8. Extended use case for the business process "Allocate cars".

USE CASE 5	Allocate cars	
Goal in Context	The pick-up branch expects to allocate cars to the rental reservations where a car is to be picked up on the following day.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has created the rental reservations.	
Success End Condition	The pick-up branch has allocated cars to the rental reservations.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger	A request by the timer to allocate cars to rental reservations.	
DESCRIPTION	**Step**	**Action**
	1	*For each rental reservation where a car is to be picked up on the following day*: allocate a car to the rental reservation (**Use Case 6**).

Table 3-9. Extended use case for the business process "Allocate a car".

USE CASE 6	Allocate a car	
Goal in Context	The pick-up branch expects to allocate to the rental reservation a car of the requested car group.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has created the rental reservation.	
Success End Condition	The pick-up branch has allocated to the rental reservation a car of the requested car group.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*There is an available car of the requested car group in the pick-up branch*: the pick-up branch allocates the car to the rental reservation and commits to provide the customer with the car.
EXTENSIONS	**Step**	**Branching Action**
	1a	*There is no available car of the requested car group in the pick-up branch*: the pick-up branch allocates to the rental reservation a car of the next higher car group (**Use Case 7**).

Table 3-10. Extended use case for the business process "Allocate a car of the next higher car group".

USE CASE 7	Allocate a car of the next higher car group	
Goal in Context	The pick-up branch expects to allocate to the rental reservation a car of the next higher car group.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has created the rental reservation and there is no available car of the requested car group in the pick-up branch.	
Success End Condition	The pick-up branch has allocated to the rental reservation a car of the next higher car group.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*There is an available car of the next higher car group in the pick-up branch*: the pick-up branch allocates the car to the rental reservation and commits to provide the customer with the car.
EXTENSIONS	**Step**	**Branching Action**
	1a	*There is no available car of the next higher car group in the pick-up branch*: the pick-up branch allocates to the rental reservation a car with bumped upgrade (**Use Case 8**).

Table 3-11. Extended use case for the business process "Allocate a car with bumped upgrade".

USE CASE 8	Allocate a car with bumped upgrade.	
Goal in Context	The pick-up branch expects to allocate to the rental reservation a car with bumped upgrade.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has created the rental reservation and there is no available car of the next higher car group in the pick-up branch.	
Success End Condition	The pick-up branch has allocated to the rental reservation a car with bumped upgrade.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*There is an available car to allocate to the rental reservation with bumped upgrade in the pick-up branch*: the pick-up branch allocates to the rental reservation a car with bumped upgrade (a car of the next higher group allocated to a rental reservation is replaced by one more higher group car, and the freed-up car of the next higher group is allocated to the rental reservation), and commits to provide the customer with the car.
EXTENSIONS	**Step**	**Branching Action**
	1a	*There is no available car to allocate to the rental reservation with bumped upgrade in the pick-up branch*: the pick-up branch allocates to the rental reservation a car of the next lower car group (**Use Case 9**).

Table 3-12. Extended use case for the business process "Allocate a car of the next lower car group".

USE CASE 9	Allocate a car of the next lower car group.	
Goal in Context	The pick-up branch expects to allocate to the rental reservation a car of the next lower car group.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has created the rental reservation and there is no available car to allocate to the rental reservation with bumped upgrade in the pick-up branch.	
Success End Condition	The pick-up branch has allocated to the rental reservation a car of the next lower car group.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*There is an available car of the next lower car group in the pick-up branch*: the pick-up branch allocates the car to the rental reservation and commits to provide the customer with the car.
EXTENSIONS	**Step**	**Branching Action**
	1a	*There is no available car of the next lower car group in the pick-up branch*: the pick-up branch allocates to the rental reservation a car that is not present in the pick-up branch (**Use Case 10**).

Table 3-13. Extended use case for the business process "Allocate a car that is not present".

USE CASE 10	Allocate a car that is not present.	
Goal in Context	The pick-up branch expects to allocate to the rental reservation a car of the requested car group that is not present (i.e., has not been returned yet).	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has created the rental reservation and there is no available car of the next lower car group in the pick-up branch.	
Success End Condition	The pick-up branch has allocated to the rental reservation a car that is not present.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*There is a suitable car of the requested car group that is not present in the pick-up branch*: the pick-up branch allocates the car to the rental reservation and commits to provide the customer with the car.
EXTENSIONS	**Step**	**Branching Action**
	1a	*There is no suitable car of the requested car group that is not present in the pick-up branch*: the pick-up branch allocates to the rental reservation a car from another branch (**Use Case 11**).

Table 3-14. Extended use case for the business process "Allocate a car from another branch".

USE CASE 11	Allocate a car from another branch.	
Goal in Context	The pick-up branch expects to allocate to the rental reservation a car transferred from a branch-proposer.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has created the rental reservation and there is no suitable car of the requested car group that is not present in the pick-up branch.	
Success End Condition	The pick-up branch has allocated to the rental reservation the car that has been transferred to the pick-up branch from a branch-proposer.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The pick-up branch sends to branch-proposers a call for proposals to transfer a car (**Use Case 12**).
	2	*Until there is timeout*: the production department waits for and receives from a branch-proposer a proposal to transfer a car (**Use Case 13**).
	3	The pick-up branch selects the cheapest (in the interests of the EU-Rent car rental company as a whole) proposal and informs the winning branch-proposer about that.
	4	The pick-up branch rejects the proposals by the losing branch-proposers (**Use Case 14**).
	5	*The pick-up branch receives the car from the winning branch-proposer*: the pick-up branch registers the car and allocates the car to the rental reservation.

Table 3-15. Extended use case for the business process "Send calls-for-proposals".

USE CASE 12	Send calls-for-proposals.	
Goal in Context	The pick-up branch expects to send to branch-proposers calls-for-proposals to transfer a car.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has created the rental reservation and there is no suitable car of the requested car group that is not present in the pick-up branch.	
Success End Condition	The pick-up branch has sent to branch-proposers calls-for-proposals to transfer a car.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each branch-proposer*: send a call-for-proposals.

Table 3-16. Extended use case for the business process "Receive a proposal".

USE CASE 13	Receive a proposal.	
Goal in Context	The pick-up branch expects to receive from a branch-proposer a proposal to transfer a car.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has sent to branch-proposers calls-for-proposals to transfer a car.	
Success End Condition	The pick-up branch has received from a branch-proposer a proposal to transfer a car.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*The pick-up branch receives from a branch-proposer a proposal to transfer a car*: the pick-up branch registers the proposal.
EXTENSIONS	**Step**	**Branching Action**
	1a	*The pick-up branch receives from a branch-proposer a refusal to transfer a car*: the pick-up branch registers the refusal.

Table 3-17. Extended use case for the business process "Inform the losers".

USE CASE 14	Inform the losers.	
Goal in Context	The pick-up branch expects to reject the proposals by the losing branch-proposers.	
Scope & Level	Pick-up branch, subfunction.	
Preconditions	The pick-up branch has selected the winning proposal.	
Success End Condition	The pick-up branch has rejected the proposals by the losing branch-proposers.	
Primary Actor **Secondary Actors**	Customer. Branch-proposer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each losing branch-proposer*: send to the branch-proposer the rejection message.

3.8. DESIGN BY EXTENDED AOR MODELLING

The modelling by goal-based use cases described in section 3.7 serves as the first step in working out the design models of a socio-technical system. We have based the design phase of the methodology proposed by us on the combination of the AOR Modelling Language (AORML), which was reviewed in section 3.4, and Object Constraint Language (OCL) of UML [OMG03a, OMG03b] which was briefly described in section 3.5. However, AORML in the form it is proposed in [Wagner00a], [Wagner01], and [Wagner03a] does not lend itself to the modelling of the motivational and functional views out of the six views of agent-oriented modelling defined in section 1.5.6. To fill in this gap, we have complemented AORML by activity diagrams which were introduced and explained in section 3.6. We call the resulting modelling language the ***extended AORML***.

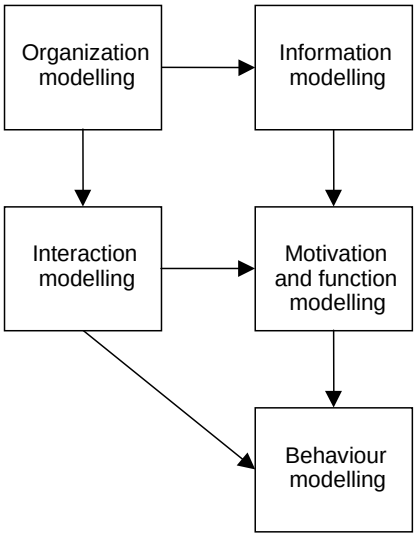


Figure 3-10. The steps of the design phase by extended AOR modelling.

The steps of the design phase and the order how they should be applied are depicted in Figure 3-10. In subsections 3.8.1 – 3.8.5, we will describe separately for each view of agent-oriented modelling how the methodology proposed by us covers it.

3.8.1. Organization Modelling

The ***organizational view*** of business modelling concerns the modelling of *active entities*, i.e. agents and agent types and relationships between them. The purposes of organization modelling are:

- to identify the organization(s) of the problem domain;
- to identify the relevant organization units that each organization to be modelled consists of;
- to identify the *roles* included by the organization units;
- to identify the types of relationships occurring between these agent types.

According to [Wagner03a], an institutional agent consists of a number of internal agents that perceive events and perform actions on behalf of it, by playing certain *roles*. Both human and artificial

agents act in roles on behalf of an organization and its units. A human agent may delegate a part or all of its rights and duties obtained through playing a role to an artificial (e.g., software) agent.

Table 2-1 may prove useful in identifying organization units and the roles included by them. However, please notice that in AORML, a ***role*** is understood as an "abstract characterization of the behaviour of a social actor within some specialized context or domain of endeavor" [Yu95a] like BranchManager (human role) or Pick-UpBranch (institutional role) in Figure 3-11, while in Role Interaction Nets also institutional agent types like Branch are referred to as roles.

In AOR Modelling, *a role is played by one or more human and/or artificial agents that act on behalf of some institutional agent of the type the role is included by* (e.g., the role BranchManager in Figure 3-11 can be played by a combination of a human and software agent that act on behalf of an agent of the type Branch).

Organization units and roles can be represented by agent diagrams of AORML where different agent types may relate to each other through the relationships of *generalization* and *aggregation*. Both organization units and roles are represented as agent types in agent diagrams. Figure 3-11 depicts the agent instance EU-Rent, representing the car rental company that belongs to the agent type Organization (not shown in the figure). The agent EU-Rent consists of instances of the internal agent types Headquarters, Branch, and AutomotiveServiceStation which form subclasses of the institutional agent type OrganizationUnit. In AORML like in UML [OMG03a], the number of instances of an agent type may be shown in the top right corner of the box with rounded corners denoting the corresponding agent type. For example, in Figure 3-11, the agent type Headquarters has just one instance, while the agent types Branch and AutomotiveServiceStation have one or more instances. An interval for the number of instances of different institutional agent types is shown in the second column of Table 2-1.

The *institutional roles* Pick-UpBranch, Drop-OffBranch, and BranchProposer form subclasses of the institutional agent type Branch. According to the definition of a role presented above, each role is valid only within a certain context. For example, the role Pick-UpBranch is valid only in the context of business processes of making a rental reservation and picking up a car. The extension of a role is thus less stable than that of its superclass.

All *human roles* represented in Figure 3-11 are subclasses of the role EmployeeOfEU-Rent, which, in turn, constitutes a subclass of the agent type Person. The instance of the agent type Headquarters includes the roles ManagingDirector and FinancialAccountingClerk. Each instance of the agent type Branch includes an instance of BranchManager, CarMaintainer, Driver, and FinancialAccountingClerk, and one or more instances of CustomerServiceClerk. Each instance of the agent type AutomotiveServiceStation includes a ServiceStationManager, one or more instances of Repairman, and a BookingClerk.

In addition to human agents, the instance of Headquarters and instances of the agent types Branch and AutomotiveServiceStation include the automated HeadquartersAgent, BranchAgent, and AutomotiveServiceStationAgent, respectively, which enable (semi)-automatic management of business processes. Additionally, instances of Branch include a simple automated agent of the type Timer.

According to [Zambonelli01], five types of relationships can be identified between the institutional agent types and/or roles. Such relationships then apply to the instances of the agent types and/or roles. Out of them, *control*, *benevolence*, and *dependency* relationships seem to be the most relevant ones to modelling interactions between agents.

Control relationships identify the authority structures within an organization. If an agent *i* controls another agent *j*, then *j* will perform any service demanded of it by *i*. For example, in Figure 3-11 there is the isSubordinateTo relationship between instances of the roles BranchManager and ServiceStationManager on one hand and the instance of the role ManagingDirector included by the Headquarters on the other hand.

Benevolence relationships identify agents with shared interests. An agent *i* is said to be benevolent to other agent *j* if *i* will offer its services to *j whenever it is able to do so*. For example, there is the isBenevolentTo relationship between instances of the roles Pick-UpBranch and Customer in Figure 3-11. This relationship typically appears between a service provider and customer. As it has been noticed in [Zambonelli01], benevolence is also the classical assumption made in research on distributed problem solving.

Dependency relationships exist between agents primarily because of resource restrictions. A dependency relationship between two agents implies a contract or agreement between the agents according to which one agent provides upon demand a part of some resource (e.g., a piece of information) managed by it to another agent. For example, the providesResourceTo relationship between the role Pick-UpBranch and the institutional agent type Headquarters in Figure 3-11 means that

a Pick-UpBranch depends on the Headquarters for information about blacklisted customers. In a similar manner, a Drop-OffBranch depends on an AutomotiveServiceStation for car servicing. Upon demand, a depender is provided with the required information or service by the dependee.

As we saw in section 2.1.4, in the ***i**** approach [Yu95a, Yu95b], more abstract task, goal, and resource dependency relationships are proposed. A *task dependency*, under which one agent specifies to another *how* the task is to be performed, but not *why*, subsumes the control relationship which means that the dependee is subordinated to the depender, as a rule. Analogously, a *goal dependency*, where the dependee is given the freedom to choose how to bring about a certain state in the world for the depender, subsumes the benevolence relationship in the sense that the dependee is assumed to be benevolent towards the depender. Finally, a *resource dependency* of ***i****, where the depender depends on the dependee for the availability of an entity (physical or informational), straightforwardly corresponds to the dependency relationship between two agents by Zambonelli [Zambonelli01].

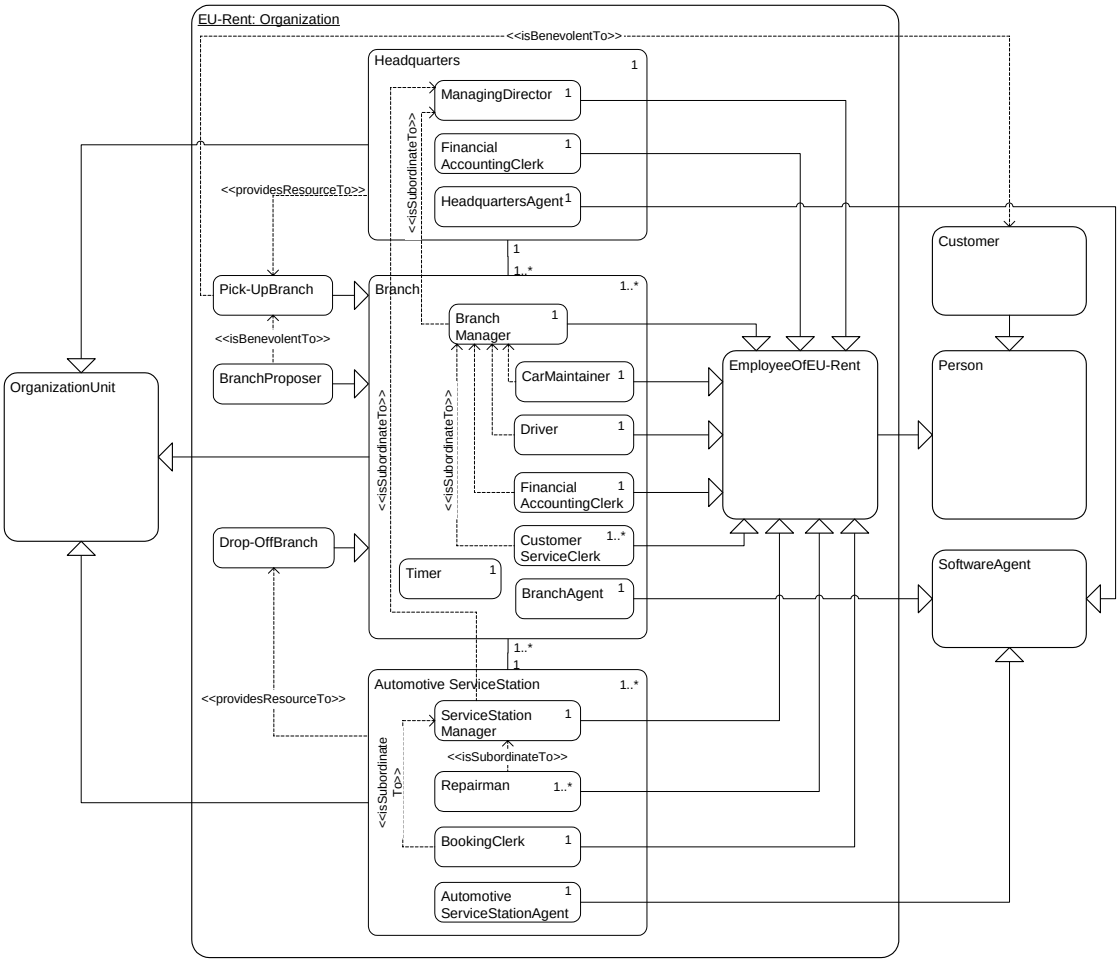


Figure 3-11. An agent diagram of the organizational view for the EU-Rent car rental company.

3.8.2. Information Modelling

The ***informational view*** of agent-oriented modelling deals with the modelling of *beliefs* of the focus agent (or agents). Each agent has beliefs about its internal agents, about its 'private' objects, and about all external agents and shared objects that are related to it.

Creation of the informational view for the focus organization(s) can be regarded as the creation of an *ontology* providing a common framework of knowledge for the agents of the organization(s) and external agents connected to the organization(s). A ***problem-oriented ontology*** is a description by truth values of the concepts and relationships of the problem domain that exist for an agent or more commonly for a community of agents [Gruber93]. Ontology consists of the *concepts* (*classes*), *relationships* between them like e.g. subsumption (inheritance), aggregation, and association, and *axioms* of the problem domain. Ontology should provide all the data structures, relationships, and axioms that are necessary for the agents for performing their actions. Ontology should also represent the agents themselves. Each agent of the problem domain can see only a part of the ontology, i.e. each agent views the ontology from a specific perspective.

According to [Corcho03], ontologies are sometimes divided into *lightweight and heavyweight ontologies*. It is claimed in [Corcho03] that "the database community, as well as the object-oriented community, also builds domain models using concepts, relations, properties, etc., but most of the times both communities impose less semantic constraints than those imposed in heavyweight ontologies". However, since ontologies created within the Business Agents' Approach include derivation rules and constraints, as a rule, they can be regarded as heavyweight ontologies. Such ontologies are created by using a combination of the principles and notations provided by the extended AORML and modified Object Constraint Language which were reviewed in sections 3.4 and 3.5, respectively

According to [Wagner98], an entity-relationship (ER) model (an object class model) of an application domain defines a formal language consisting of:

- the finite set of application domain predicates established by the entity-relationship analysis;
- the set of all possible attribute values, i.e. the union of all attribute domains, which forms the set of constant symbols.

The ontology to be created within the informational view can thus be treated as the *finite set of domain predicates* which were introduced in section 3.3. As we saw in section 3.3, there are three kinds of basic domain predicates: *informational entity types*, *relationship types*, and *attributes*. For representing them, we use agent diagrams of AORML which were briefly described in section 3.4.4. Figure 3-13 depicts an agent diagram of the informational view of the domain of the EU-Rent car rental company. As is shown in Figure 3-13, the car rental company EU-Rent is the focus agent of the type Organization. It consists of instances of the institutional agent types Headquarters, Branch, and AutomotiveServiceStation.

Please note that *agent diagrams of the informational view can be created with different levels of preciseness and different internal agents represented*. It is thus not necessary to represent within the informational view all the internal agent types and instances of EU-Rent that were modelled within the organizational view in Figure 3-11.

In Figure 3-13, the agent EU-Rent has customers as external human agents of the type Customer, which is a subtype of Person, and employees as internal human agents of another subtype of Person – EmployeeOfEU-Rent. In general, institutional agents may also serve as customers, but this possibility is not reflected by Figure 3-13.

According to the associations between informational entity types shown in the agent diagram of Figure 3-13, the EU-Rent car rental company has zero or more customers. A Customer has zero or more instances of RentalOrder/Invoice, to each of which may have been allocated at most one RentalCar. The object type RentalCar forms a subclass of the more general object type Car. Each instance of RentalCar belongs to a specific CarGroup, and each instance of CarGroup has zero or more instances of RentalCar. Instances of CarGroup of different rank (and price) are connected to each other by the association that is navigated in the directions denoted by the nextLowerGroup and nextHigherGroup *rolenames* [OMG03a]. As a rental order always specifies the group of the car requested or provided, there is an instance of CarGroup associated with each RentalOrder.

The meanings of the *attributes* of informational entity types, such as **personID**, birthDate, and age of Customer, as well as carID of RentalCar and mileage of Car in Figure 3-13 are self-evident. We distinguish *identifier attributes*, like **rentalOrderID**, by presenting them in bold as is shown in Figure 3-13. ***Identifier attributes*** are used to identify the entities in agent messages as is demonstrated in

section 3.8.3. An identifier attribute is implicitly assigned with a unique value when the corresponding entity instance is created.

3.8.2.1. Modelling of Derivation Rules

We distinguish three kinds of business rules of the derivation rule type: *derived attributes*, *status predicates*, and *intensional predicates*. Strictly speaking, *derived informational entity types*, like those derived by means of the inheritance relationship, and derived associations also form kinds of derivation rules. However, we confine our treatment of derivation rules to derived attributes, status predicates, and intensional predicates.

Derived attributes are the attributes of an entity whose values are computed or inferred from the values of the entity's other attributes. In [Wagner98] such attributes, i.e. the attributes whose values are not explicitly stored in the VKB of an agent but rather have to be computed in some way, are termed ***attribute functions***. For example, the value of the attribute age of an instance of Customer is computed from the present date, returned by the calendar function now(), and the value of the attribute birthDate of the Customer. Analogously, the value of the attribute mileageSinceService of a RentalCar is computed by subtracting from the value of the attribute mileage the value of the attribute mileageAtService which is registered before every regular maintenance of a car.

Status predicates represent the *statuses* of instances of informational entity types like, for example, isBlacklisted and hasCar of a Customer, isReserved and isDroppedOff of a RentalOrder, and isPresent and isInService of a RentalCar. Status predicates do not take parameters of any kind. A status predicate can be expressed with a Boolean-valued attribute.

Intensional predicates are in our approach derived Boolean-valued attributes[9]. Intensional predicates take parameters. An example of an intensional predicate in Figure 3-13 is hasCapacity(Date, Date) of the object type CarGroup which checks whether the given car group has additional rental capacity between the pick-up time and drop-off time requested.

Derived attributes and status and intensional predicates form subtypes of domain predicates. They all are defined by derivation rules, as is shown in the metamodel of the Business Agents Approach' in Figure 3-3. The modelling technique proposed by us includes representing derivation rules using the Object Constraint Language (OCL) which is now a part of UML [OMG03a]. Originally in [Taveter01c], we also sketch a notation for visualizing derivation rules of some kind.

Derived attributes are expressed in our approach by using invariants of OCL. For example, the derivation of the value of the attribute age of an instance of Person from the present date, returned by the calendar function now(), and from the value of the attribute birthDate of the Person, can be expressed as follows:

```
context Person inv:
self.age = now() – self.birthDate
```

Analogously, the derivation of the value of the attribute mileageSinceService from the values of the attributes mileage and mileageAtService of an instance of RentalCar can be expressed as the following invariant:

```
context RentalCar inv:
self.mileageSinceService = self.mileage – self.mileageAtService
```

As we have argued in [Taveter01c], derivation rules of the type *status predicates* can be expressed both graphically and in OCL. Consider the rule D1: *A car is available for rental if it is physically present, is not assigned to any rental order, is not scheduled for service, and does not require service*. D1 can be formalized as a logic programming-style derivation rule in the following form:

```
RentalCar.isAvailable(x) ←
RentalCar.isPresent(x)
∧¬∃y(isAssignedTo(x, y))
∧¬RentalCar.requiresService(x)
∧¬RentalCar.isScheduledForService(x)
```

where the variable x refers to the identifier of a car.

The rule is applicable to an instance of the conclusion class if the conjunction of the conditions evaluates to true. Each condition is either an atom (in the sense of logic programming terminology), a

[9] In general, the term "intensional predicate" just means a predicate that is defined by means of a derivation rule instead of e.g. presenting a set of extensional facts.

negated atom, or a negated existentially qualified atom where all free variables occur also among the variables of some atom. An example of the latter condition type is the expression ¬∃y(isAssignedTo(x, y)).

The condition and conclusion part of certain types of derivation rules can be expressed in OCL. Since there is no genuine rule conditional in OCL but only the Boolean implication operator **implies**, and the semantics of OCL does not include a proper treatment of derivation rules[10], we have to form a pseudo-OCL expression where the **IF** operator represents the rule conditional ←.

```
context RentalCar inv:
self.isAvailable IF
self.isPresent
and self.RentalOrder->isEmpty()
and not self.requiresService
and not self.isScheduledForService
```

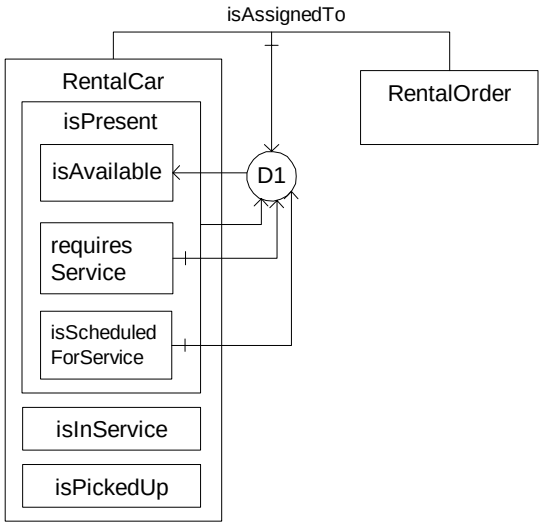


Figure 3-12. Visualizing the derivation rule D1.

There has been some work on the visualization of logic programs [Dewar91, Brayshaw91, Neufeld97], focusing mainly on the visualization of proof trees and the control flow in an AND/OR tree by displaying the success or failure of rules and the associated unification process. These works are motivated by the desire to support the debugging, and the execution analysis, of logic programs. We have visualized the status predicate D1 in Figure 3-12 in a different way. In this graphical rendering of derivation rules, the incoming arrows form the conditions of the rule, and the outgoing arrow represents the conclusion (referring to instances of some informational entity type). A negated atom of the condition is represented by crossing the source part of the arrow denoting the condition. However, in Appendix C we represent status predicates as well as other derivation rules of the domain of the EU-Rent car rental company in OCL extended by the **IF** operator.

Finally, we express derivation rules of the type *intensional predicates* as operations of OCL. For example, the intensional predicate hasCapacity(Date, Date) in Figure 3-13 can be expressed as the operation below attached to the object type CarGroup. The *helper operation* hasOverlappingRentalOrder according to the new version of OCL [OMG03b] is defined in Appendix C.

```
context CarGroup::hasCapacity(pickUpTime: Date, dropOffTime: Date): Boolean
post: result = (self.rentalCar->exists(not(isScheduledForService) and
not(hasOverlappingRentalOrder(pt, dt))))
```

3.8.2.2. Modelling of Integrity Constraints

For representing business rules of the type ***integrity constraints***, we again make use of OCL. Our example of the business rule stating that a customer of EU-Rent must be at least 25 years old corresponds to a simple state constraint that can be expressed in OCL in the following way:

```
context Customer inv:
```

[10] As has been noticed in logic programming semantics, because of their nonmonotonic nature, derivation rules do not correspond to Boolean implication formulas.

self.age > 25

where the OCL keyword **inv** indicates an invariant.

A simple constraint like the one presented above can be included as an attribute constraint in the attribute list of a type, as suggested by UML. There are other constraints, such as multiplicity constraints, for which a visualization is readily available. In the general case, however, since constraints correspond to logical sentences of a formal language, there may be no straightforward way to visualize a constraint, and we have to be content with a textual representation. A comprehensive but somewhat complicated technique for visualizing constraints – Ross Notation – was presented and evaluated in section 2.1.1.

3.8.2.3. Extensions to Agent Diagrams

External AOR models represent agent types and/or agents and object types of the problem domain under inspection. The structure of beliefs of each focus agent about the entities that it is associated with by default corresponds to the structure of these informational entity types. For example, since the rectangle representing the agent type Customer is connected to the rectangle of the object type RentalOrder/Invoice in Figure 3-13, agents of this type have about instances of RentalOrder/Invoice beliefs of the structure depicted in the figure. However, sometimes a belief of an agent needs a different representation, using different attributes and status and intensional predicates, because it does not directly correspond to the structure of an "external" object or agent but rather to the agent's "personal" view of it. For such a case, we are extending the AOR Modelling Language by the dependency arrow with the stereotype <<represents>> between the internal representation and the corresponding external informational entity type. For example, as is shown in the agent diagram of Figure 3-13, there is an internal representation of the object type CarGroup within agents of the type Branch.

Wagner03aWagner03aIn the agent diagram depicted in Figure 3-13, the agent EU-Rent has specific internal representations of the object types RentalCar and RentalOrder/Invoice. The first of them is shared between the EU-Rent's internal agents of the types Branch and AutomotiveServiceStation, while the second one is shared between instances of the agent types Headquarters and Branch. The corresponding dependency arrows with the stereotype <<represents>> from the internal RentalCar and RentalOrder/Invoice rectangles within EU-Rent to the corresponding external rectangles indicate that the car rental company needs to represent information about instances of RentalCar and RentalOrder/Invoice using additional attributes, such as mileageAtService and serviceStartTime, additional status predicates like isPresent and isPaid, and additional intensional predicates, such as isAvailableOfOwnGroup(RentalOrder) and canBeReAllocated(RentalOrder) which have to do with the allocation of cars to rental orders. Analogously, the internal representation of the object type CarGroup within the agent type Branch includes the additional status predicate hasCapacity(Date, Date). The latter is expressed by the dependency arrow with the stereotype <<represents>> from the internal CarGroup rectangle of Branch to the corresponding external rectangle.

Analogously to object types, if another agent type is to be represented by a focus agent (type) with "proprietary" attributes or status predicates (that only have meaning for the focus agent), such as when a Customer is to be represented by the Headquarters with the proprietary status predicates isBlacklisted and hasCar, then the corresponding agent type rectangle with an agent type name is drawn as a representation of the "external" agent type within the focus agent (type), as can be seen in Figure 3-13.

As it is formulated in [Wagner03a], if an object type belongs exclusively to one agent or agent type (in the sense of a UML *component* class), the corresponding rectangle is drawn inside of this agent (type) rectangle. Otherwise, if the object type rectangle is drawn outside of the respective agent (type) rectangles, the focus agents have by default beliefs of the corresponding structure about its instances. However, it is emphasized in [Wagner03a] that *an "external" object type does not imply that all the agents connected by an association to it have the same beliefs about it, or, in other words, that there is a common extension of it shared by all agents*. For example, different instances of the agent type Customer in Figure 3-13 all hold different sets of instances of RentalOrder/Invoice in their VKB's, as a rule. Since the extensional difference between internal representations is implicit, as was stated above, the internal representations of the object type RentalOrder/Invoice within the agent type Customer are not shown in Figure 3-13.In addition to the way generalization is visualized in ER diagrams and UML class diagrams, by means of a special arrow, a *subclass* can also be visualized as a rectangle within its superclass, following [Harel87]. Since all entities of some type that have a certain status (satisfy a certain status predicate) form a subclass of the informational entity type (see, e.g. [IDEF94]), we use the notation for subclasses also for representing entities having a certain status where it seems to be

especially useful in terms of visual clarity. For example, in the internal view by EU-Rent, an object of the type RentalCar in Figure 3-13 can have the status isPresent, isPickedUp, or isInService. A status predicate may include substatus predicates. For example, an instance of RentalCar in Figure 3-13 with the status isPresent has the substatus requiresService, isScheduledForService, or isAvailable.

Please notice that a graphical box inside of another in agent diagrams of AOR modelling has one of three different meanings: isBeliefOf, isSubclassOf (inheritance which includes status predicates), and isComponentOf (aggregation). It is easy to distinguish the first case from the others because the rectangle representing an entity (type) of an agent's "private" belief structure is located inside of the rectangle representing the corresponding agent (type). It is more complicated to distinguish between aggregation and inheritance relationships which are context-dependent until the issue will be settled in the forthcoming AORML Reference Manual[11]. However, in the extended AOR models included in this thesis we have represented the isSubclassOf relationship by using a UML inheritance arrow, like between Customer and Person in Figure 3-13, and have reserved the controversial notation for representing status predicates, like the status predicate isPreliminary included by the object type RentalOrder/Invoice in Figure 3-13.

[11] According to the personal communication with the author of the original AORML.

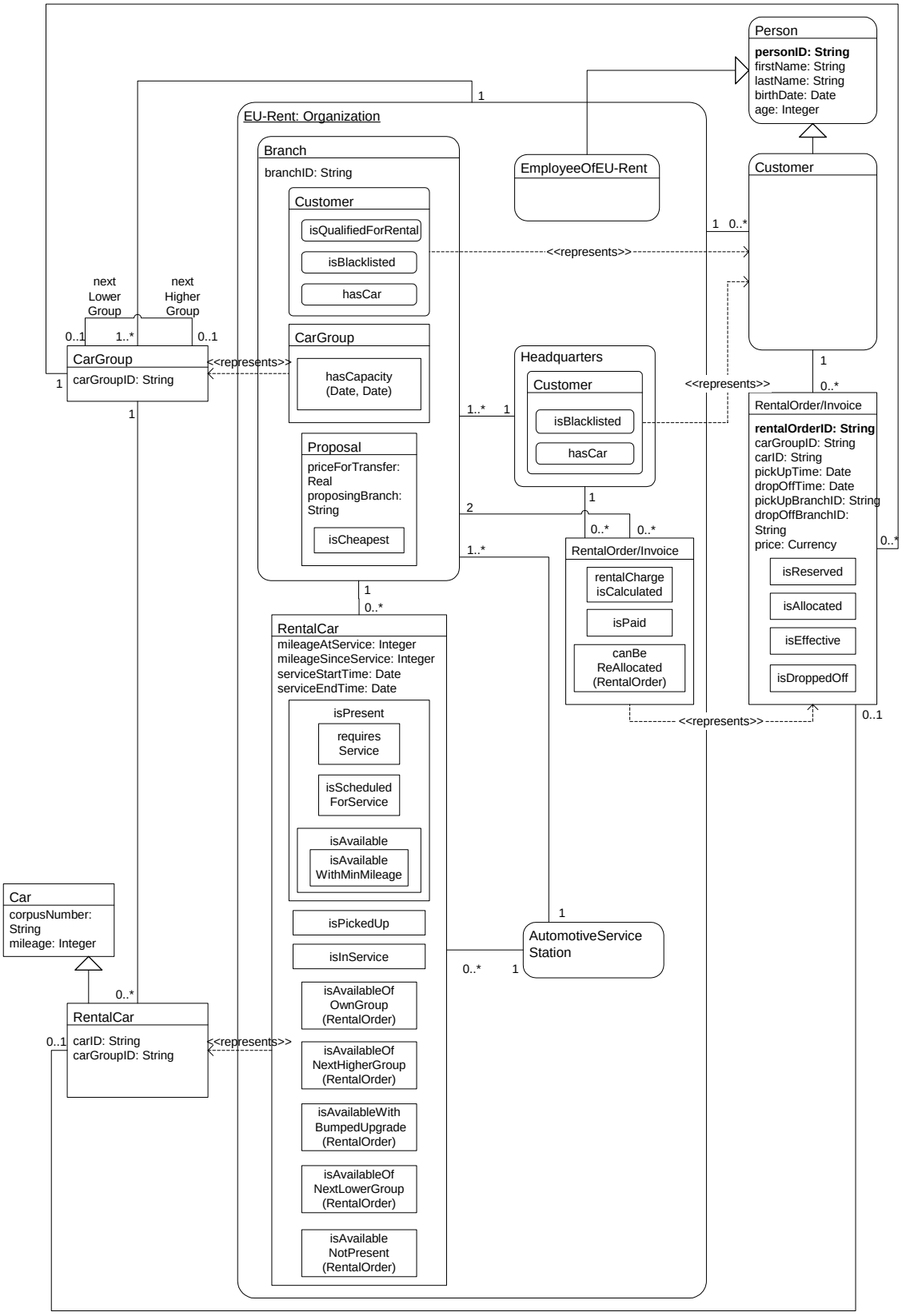


Figure 3-13. An agent diagram of the informational view for the EU-Rent car rental company.

3.8.3. Interaction Modelling

The ***interactional view*** concerns the modelling of *interactions* and *communication* between the agents. It is represented by using *interaction frame diagrams* which were introduced in section 3.4.4. An interaction frame between two agent types consists of those action event types and commitment/claim types that form the basis of the interaction processes in which these two agent types are involved. Unlike a UML sequence diagram, it does not model any sequential process but provides a static picture of the possible interactions and evolvement of commitments/claims between the agent types.

Figure 3-16 depicts the interaction frames between the agent types Customer, Branch, Headquarters, and AutomotiveServiceStation. The interaction frames are formed in accordance with the control, benevolence, and dependency relationships that were identified between the institutional agent types and/or roles in section 3.8.1. A *control* relationship between an agent requesting for some service and the agent providing the service means that the service provider agent can not refuse the service requested from it. For example, in Figure 3-11 there is the isSubordinateTo relationship between instances of the roles BranchManager of Branch and ManagingDirector of Headquarters. Since, as we argued in section 3.8.1, internal agents included by an institutional agent act on behalf of the institutional agent, this control relationship also applies to the institutional agents of the respective types Branch and Headquarters. Under a *benevolence* relationship between a service requester and the service provider, the service provider performs the service requested if it is able to do so, but the service provider also has an option to refuse the service requested. This can, for example, happen on the insufficiency of the resources required. For example, there is the isBenevolentTo relationship between instances of Pick-UpBranch and Customer in Figure 3-11. Finally, since a *dependency* (for a resource) relationship, like the relationship between instances of Drop-OffBranch and AutomotiveServiceStation in Figure 3-11, implies a valid contract or agreement between the parties to provide the service requester with the service, a service request is usually not refused under this kind of dependency.

Notice that *not* for all action event types in Figure 3-16, there is a corresponding commitment/claim type. For instance, there are no commitments of (or claims against) customers to pick up a car, whereas there are commitments and claims to return a car. Notice also that even though a commitment of a Customer of the type returnCar in Figure 3-16 is towards an instance of the agent role Drop-OffBranch, this commitment is also discharged if the Customer returns the car to a branch other than the agreed drop-off branch (true, he/she is charged a drop-off penalty then according to a business rule of the EU-Rent car rental company).

In principle, each communicative action event type in Figure 3-16 could be modelled as a separate object type consisting of the message type name and a number of attributes. For example, the first message type request provideCar in Figure 3-16 could be modelled as the object type CarReservationRequest. This way, we would obtain many domain-specific message languages. Such an approach has been followed in EDIFACT [Salminen95] and in the emerging standards for e-business RosettaNet [RosettaNet] of electronics industry and PapiNet [PapiNet] of paper industry. However, within each domain this will eventually lead to an exponential explosion of message types which should be avoided. To avoid such explosion, we have chosen to model communicative action event types as types of *speech act messages* [Austin62] where different modalities in the form of speech acts can be applied to the same proposition. A speech act message has the mandatory form *m(c)* where *m* is the message type (like *request*, *query-if*, *inform*, etc., expressing the 'illocutionary force' according to [Searle85]), and *c* is the message content, composed of propositions and/or action terms. We define a ***proposition*** as a logical sentence referring to one or more domain predicates of the problem domain's ontology. An ***action term*** identifies the type of action that, for example, one agent requests another agent to perform. We elaborate on AORML as it was described in sections 3.4.1 through 3.4.4 by stating that ***each action term refers to some non-communicative action event type***.

Agent communication based on speech acts contrasts with the language used in the literature on object-oriented programming, where objects 'communicate' or 'interact' with each other by sending 'messages'. As we saw above, a speech act message consists of the message type and content, while an OO message has no generic structure at all. Notice also that the UML [OMG03a] term 'collaboration' between objects corresponds only to a very low-level sense of communication and interaction. In fact, sending a 'message' in the sense of OO programming corresponds rather to a (possibly remote) procedure call, and not to a communication act (or speech act). Object-orientation thus does not capture communication and interaction in the high-level sense of business processes carried out by business agents.

3.8.3.1. Representation of Action Event Types

In interaction frame and activity diagrams, non-communicative action event types are represented as combinations of action terms and propositions. The constructs for representing them are defined by the activity modelling language which is presented in Appendix B. For example, in Figure 3-16 the action term provideCar is used in combination with the proposition RentalCar(corpusNumber(?String1) mileage(?Integer) carID(?String2) carGroupID(?String3)), which includes the domain predicate RentalCar, representing an object type, and the domain predicates corpusNumber, mileage, carID, and carGroupID, representing attributes of the object type. When the action event type is instantiated by e.g. a simulation environment, to the terms (variables) of the proposition ?String1, ?Integer, ?String2, and ?String3 are assigned values of the corresponding types. As a result, the instantiated proposition refers to the representation of a physical object, such as a rental car, that is transferred from one agent to another by the action event. As commitment/claim types may be coupled with action event types, propositions may be associated with them in exactly the same manner as with action event types.

Analogously, within a communicative action event (i.e., an agent message), an instance of an informational entity type can be represented by means of an instantiated proposition that includes one or more domain predicates of the ontology in the way demonstrated by the following example:

```
(RentalOrder
    : rentalOrderID "245"
    : carGroupID "B"
    : carID "764 WGY"
    : pickUpTime 0106041200
    : dropOffTime 0706041730
    : pickUpBranchID "Tallinn Airport"
    : dropOffBranchID "Riga Centre"))
```

In interaction frame and activity diagrams, we represent the terms (variables) standing for instances of object types and of internal representations of agent types as ?EntityType. Analogously, the terms standing for instances of datatypes are abbreviated as ?DataType. An object of the type RentalOrder and a data item of the type String are thus represented as ?RentalOrder and ?String, respectively. If there is more than one data item of the same type within an action event, its representation within the action event type is followed by the number of order, like ?String1. However, in some cases it is sufficient to refer to an instance of an informational entity type by using just the value of the instance's *identifier attribute*. We thus define the construct EntityType(?DataType) that enables to refer to an instance of EntityType by the value of its identifier attribute of the type DataType. For example, the instantiated proposition RentalOrder("245") refers to the same instance of RentalOrder as the proposition in the example above by making use of the RentalOrder's identifier attribute rentalOrderID of the type String.

As a rule, the mental effect co-occurring with a non-communicative action event evaluates one or more domain predicates of the problem domain's ontology which is expressed in the form of an agent diagram in our approach. For example, the mental effect accompanying the occurrence of an action event of the type pickupCar in Figure 3-16 evaluates the status predicate isEffective of the corresponding instance of RentalOrder.

3.8.3.2. Introducing achieve-*Construct Type*

In addition to requesting another agent to perform an atomic action, one agent can directly request another agent to make true some proposition expressed in terms of domain predicates of the ontology. This modelling solution is used when there are no action events that would serve as 'carriers' of the request, such as action events of the type provideCar. For example, in Figure 3-16 a request by a Branch to service a car is expressed as a request to make true the proposition consisting of the status predicate isInService applied to the corresponding instance of RentalCar because there is no action event type for servicing a car in the problem domain of car rental. In a similar manner, there is no action event type for scheduling and performing a production activity by a resource unit (v. section 4.1.5.3) or for performing an advertising campaign by a media agency (v. section 4.2.4.2). Making some proposition true involves the occurrence of several atomic communicative and non-communicative action events between the respective agents, as a rule.

For modelling situations as the ones described above, we are introducing an achieve modelling construct type to denote achieving, i.e. making true, some proposition which is defined in terms of the domain predicates specified by the informational view of agent-oriented modelling. An achieve construct type may be coupled with the corresponding *see-to-it-that* (*stit*)-commitment/claim type. An

achieve construct type and the *stit*-commitment/claim type coupled with it are visualized like an action event type and the commitment/claim type coupled with it but drawn with a thick line. For example, Figure 3-16 depicts the achieve construct type achieve(isAllocated(?RentalOrder)) which is coupled with the corresponding *stit*-commitment/claim type achieve(isAllocated(?RentalOrder) ?Date).

3.8.3.3. Interaction Ontology

In order to be able to communicate and interact, in addition to sharing common object types defined in the informational view, the agents of the problem domain should have a common understanding of the communicative and non-communicative action event types referred to by them, as well as of the types of the corresponding commitments/claims formed between the agents. Therefore, we extend the set of shared object types explained in section 3.8.2 by the set of communicative and non-communicative action event types and commitment/claim types coupled with them. As was described in section 3.4.5, to all action event types is applied the stereotype ActionEvent. It is shown in Figure 3-14 that the stereotype ActionEvent has the subclasses CommunicativeActionEvent and NonCommunicativeActionEvent.

As is depicted in Figure 3-15, each communicative action event type is represented as an entity type to which is applied the stereotype CommunicativeActionEvent. When the stereotype CommunicativeActionEvent is applied to a communicative action event type, the attribute performative defined for the stereotype contains the name of the communicative action event type, like "request" in the example of Figure 3-15. In addition, all communicative action event types extend the abstract object class *CommunicativeActionEventType* that defines the attributes senderID, receiverID, and content of the type String. According to [OMG03a], abstract classes may not be directly instantiated and exist only for other classes to inherit and reuse the features declared by them, like the attributes defined by the abstract class *CommunicativeActionEventType*. When a communicative action event type is instantiated, the attributes senderID and receiverID contain the values of the identifier attributes of the sender and receiver agent, respectively, while the attribute content holds the message content, composed of instantiated propositions and/or action terms. This means that we view a communicative action event as a structured document in the same way as a message is understood in EDIFACT [Salminen95], FIPA ACL [ACL97], RosettaNet [RosettaNet], and PapiNet [PapiNet].

Each non-communicative action event type, such as provideCar in Figure 3-15, is also represented as an entity type of the same name to which is applied the stereotype NonCommunicativeActionEvent. In addition, all non-communicative action event types extend the abstract object class *NonCommunicativeActionEventType* that defines the attributes sourceID and targetID of the type String and the attribute about of the type OclAny. When a non-communicative action event type is instantiated, the attributes sourceID and targetID contain the values of the identifier attributes of the agents that respectively perform and perceive an action of the corresponding type. The attribute about of the type OclAny, defined by OCL [OMG03a], specifies the entity that the instantiated non-communicative action event refers to. For example, the attribute about of a non-communicative action event of the type provideCar modelled in Figure 3-15 specifies the instance of RentalCar that is delivered by the Branch to the Customer.

Analogously to an action event type, a commitment/claim type is represented as an entity type to which is applied the stereotype CommitmentClaim. It is shown in Figure 3-14 that the stereotype CommitmentClaim is divided into the stereotypes ToDoCommitmentClaim and STITCommitmentClaim. As was explained in section 3.4.5, the stereotype ToDoCommitmentClaim defines the attribute actionEventTypeName of the type String. When the stereotype is applied to a *to-do*-commitment/claim type, this attribute contains the name of the action event type that the commitment/claim type is coupled with, like "provideCar" in the example of Figure 3-15. All commitment/claim types extend the abstract object class *CommitmentClaimType* depicted in Figure 3-15 that defines the attributes dueTime, sourceID, and targetID of the respective types Date, String, and String. When a commitment/claim type is instantiated, these attributes specify the due time of the commitment/claim and the identifiers of the agents that the commitment/claim occurs between. In addition, the abstract object class *ToDoCommitmentClaimType* shown in Figure 3-15 defines the attribute about of the type OclAny which specifies for an instance of a *to-do*-commitment/claim type the entity that the commitment/claim is about. Analogously, the abstract object class *STITCommitmentClaimType* shown in Figure 3-15 defines the attribute achieve of the type OclExpression specifying, when a *stit*-commitment/claim type is instantiated, the proposition that the commitment/claim is about. The type OclExpression is the logical expression type defined by OCL [OMG03a]. Please notice that a *stit*-commitment/claim type as an anonymous class [OMG03b] is determined by the type of the proposition included by it.

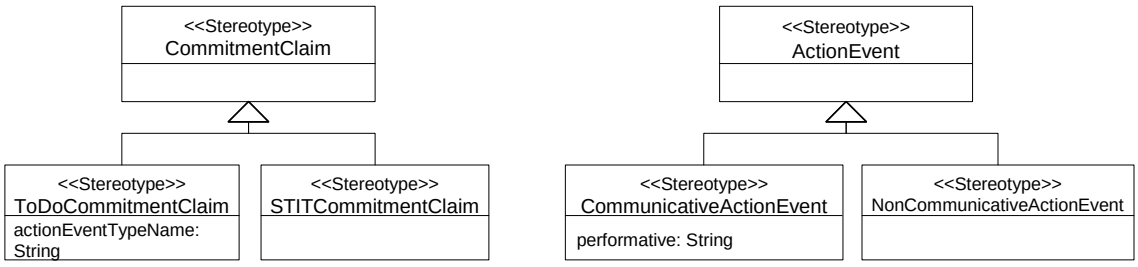


Figure 3-14. The hierarchy of stereotypes.

The beliefs of an agent contain knowledge about action events that have been performed or perceived by the agent and about commitments/claims that the agent is involved in. Instances of the types of *to-do*-commitments/claims and *stit*-commitments/claims represent in the agent's VKB the respective commitments/claims in force. When the action event that discharges a commitment/claim occurs, the representations of the corresponding commitment/claim are deleted from the VKB's of one or several agents involved, and the action event that has occurred is recorded in their VKB's as an instance of the corresponding action event type. Since a *stit*-commitment/claim type is viewed as an anonymous class [OMG03b], it can be referred to by navigating to it from an instance of one of the agent types it occurs between in the direction of the rolename stitCommitmentClaim, as is shown in Figure 3-15. The originating agent instance can be the contextual agent instance, which is referred to by self, as in the examples of creating and deleting commitments/claims presented in section 3.8.4.

In addition to storing non-communicative action events, the interaction ontology represented in Figure 3-15 provides an agent with the structure required to "remember" communicative action events that the agent has created or perceived. A "memory" of such a kind can be used for learning by the agent.

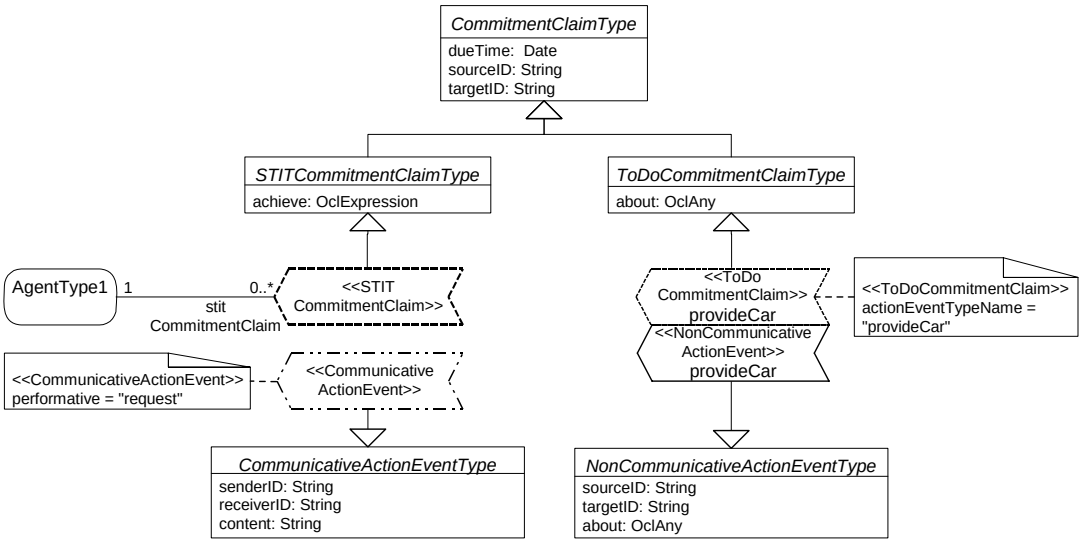


Figure 3-15. The interaction ontology.

3.8.3.4. Example of an Interaction Frame Diagram

When the agent message type request provideCar(?String1 ?Date1 ?Date2 ?String2) represented in Figure 3-16 is instantiated, the data items of the types String and Date in the message contain the identifier of the car group, the requested pick-up time and drop-off time, and the identifier of the drop-off branch, respectively. Besides, the interaction frame between the agent types Customer and Branch shown in Figure 3-16 includes the achieve construct type achieve(isAllocated(?RentalOrder)), which is coupled with the corresponding *stit*-commitment/claim type, the communicative action event types refuse provideCar(?String1 ?Date1 ?Date2 ?String2), agree achieve(isAllocated(?RentalOrder)), request provideCar (RentalOrder(?String)), and agree provideCar(?RentalCar) (or refuse provideCar(?RentalCar)), and the non-communicative action event types pickupCar, provideCar, and returnCar. The last two action event types are coupled with the corresponding commitment/claim types. Notice that the Customer is informed

about the instance of RentalOrder only when and if the rental reservation request has been accepted by the Pick-UpBranch.

The interaction frame between the agent types Branch and Headquarters depicted in Figure 3-16 consists of the following pairs (protocols) of agent message types:

- query-if(isBlacklisted(Customer(?String))) and inform([not]isBlacklisted(Customer(?String)));
- query-if(hasCar(Customer(?String))) and inform([not]hasCar(Customer(?String)));
- query-ref(RentalOrder(?String)) and inform(?RentalOrder).

The square brackets enclosing the not keyword in the first two protocols stand for optionality. With the third protocol, the drop-off branch asks for and receives the required instance of RentalOrder.

In addition, the interaction frame between the agent types Branch and Headquarters includes the standalone message types inform(isEffective(?RentalOrder)) and inform(isDroppedOff(RentalOrder(?String))). By using their instances, a Branch provides the Headquarters with an effective rental order and informs the Headquarters that the car in the rental order has been dropped off, respectively.

Finally, the interaction frame between the agent types Branch and AutomotiveServiceStation consists of the agent message type request achieve(isServiced(?RentalCar)) and the achieve construct type achieve(isServiced(?RentalCar)) which is coupled with the corresponding *stit*-commitment/claim type.

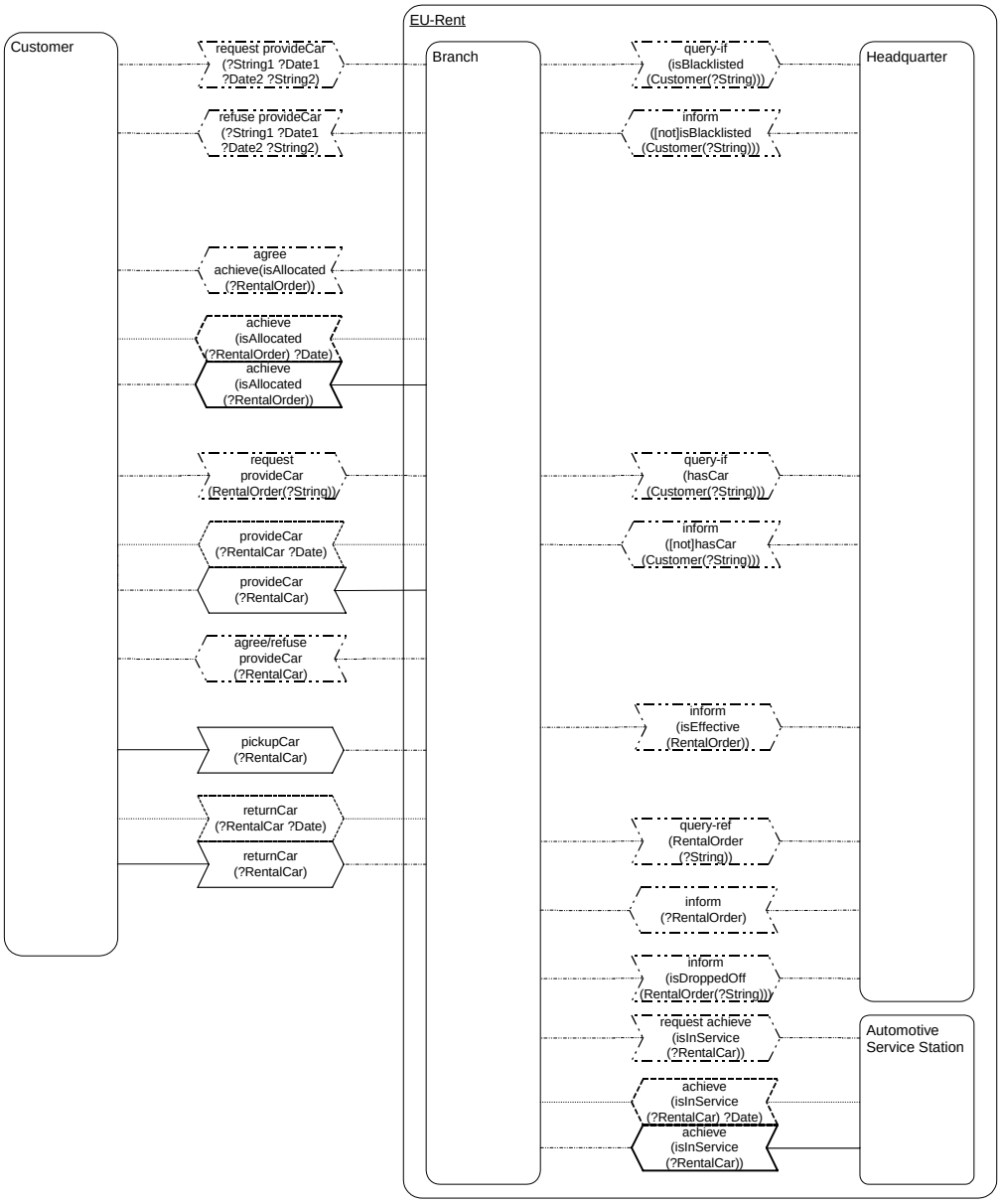


Figure 3-16. The extended *interaction frame diagram* for the EU-Rent car rental company.

3.8.4. Function and Motivation Modelling

The **functional view** of agent-oriented modelling deals with the modelling of *activities* performed by agents. It enables to specify *what* has to be done. This view is closely related to the **motivational view** which deals with the modelling of the *goals* the agents are trying to achieve, because the goals are attached to the activities. All the activity types defined at the design step of function modelling are *unspecified*. This means that we declare for each activity type its name, optional input parameters, an optional precondition, and an optional goal, but we do not specify the actions included by it. In other words, we model activity types in terms of *control flow*, *data flow*, *preconditions*, and *goals*.

3.8.4.1. Describing Activity Types

Within the functional view of the Business Agents' Approach, activity types are extracted from goal-based use cases and described by activity diagrams according to the following recursive procedure:

1. Turn the main scenario of the primary task into an activity type of the agent (type) in focus.
2. Turn the trigger of the main scenario into the reaction rule starting an activity of the corresponding type of the focus agent in response to perceiving the action event created by an external agent of the type corresponding to the primary actor.
3. Set the main scenario as the current scenario.
4. For the next step of the current scenario, if there are more steps left:
    - if the step does not include any subordinate use cases, turn the step into a sequential elementary activity type, connect it to the previous sequential elementary activity type, and return to 4;
    - if the step includes a subordinate use case:
        - turn the subordinate use case into a subactivity type;
        - if the composition of the subordinate use case is to be modelled, set the main scenario of the subordinate use case as the current scenario, and return to 4;
    - if the step includes performing an action or making true a proposition directed towards the primary or secondary actor of the use case, draw an action event or *achieve*-construct rectangle, possibly coupled with the corresponding commitment/claim type rectangle, between the corresponding activity type rectangle and external agent (type) rectangle;
    - if the step includes an internal state change, draw a mental effect arrow from the corresponding activity type rectangle to the relevant object type, status predicate, or commitment/claim type rectangle, or the association line.
5. Return to the enclosing scenario and return to 4, or finish, if there is no enclosing scenario.

At the design step of function modelling, just the main success scenarios are extracted from goal-based use cases and modelled. The extension scenarios will be dealt with only during the design step of behaviour modelling to be described in section 3.8.5. Also, the <*time or sequence factor*> and <*condition*> elements of use case steps are ignored until the design step of behaviour modelling. In the example of the car rental company, the main scenario of the primary task "Have a car reserved" presented in Table 3-4 is modelled in the activity diagram of Figure 3-17 as the activity type "Manage car reservation" of the institutional agent role Pick-UpBranch. In the same way, the main scenario of the primary task "Pick up the car" described in Table 3-5 is represented in Figure 3-17 as the activity type "Manage pick-up". The subfunctions "Check the customer for blacklistedness" and "Allocate cars" of the use case "Have a car reserved", which are presented in Tables 3-6 and 3-8, respectively, are modelled in Figure 3-17 as the corresponding subactivity types of the same names. The subactivity type "Check the customer for blacklistedness" is modelled with a hidden composition, while the subactivity type "Allocate cars" includes the elementary subactivity type "Allocate a car", in accordance with the use cases presented in Tables 3-8 and 3-9. The subfunction "Check the customer for another car" of the use case "Pick up the car", which is presented in Table 3-7, is also modelled in the activity diagram of Figure 3-17 as the corresponding activity type of the same name. The other steps of the use cases "Have a car reserved" and "Pick up the car" are represented as the following elementary activity types of the focus agent type: "Create rental reservation", "Check qualifications", "Hand over the car", and "Inform the Headquarters on pick-up". While processing the use case "Have a car reserved" in Table 3-4, the <*time or sequence factor*> element "*It is the end of the day*" in step 3 of the use case is ignored. Analogously, the <*condition*> element "*For each rental reservation…*" in step 1 of the use case "Allocate cars" in Table 3-8 is ignored.

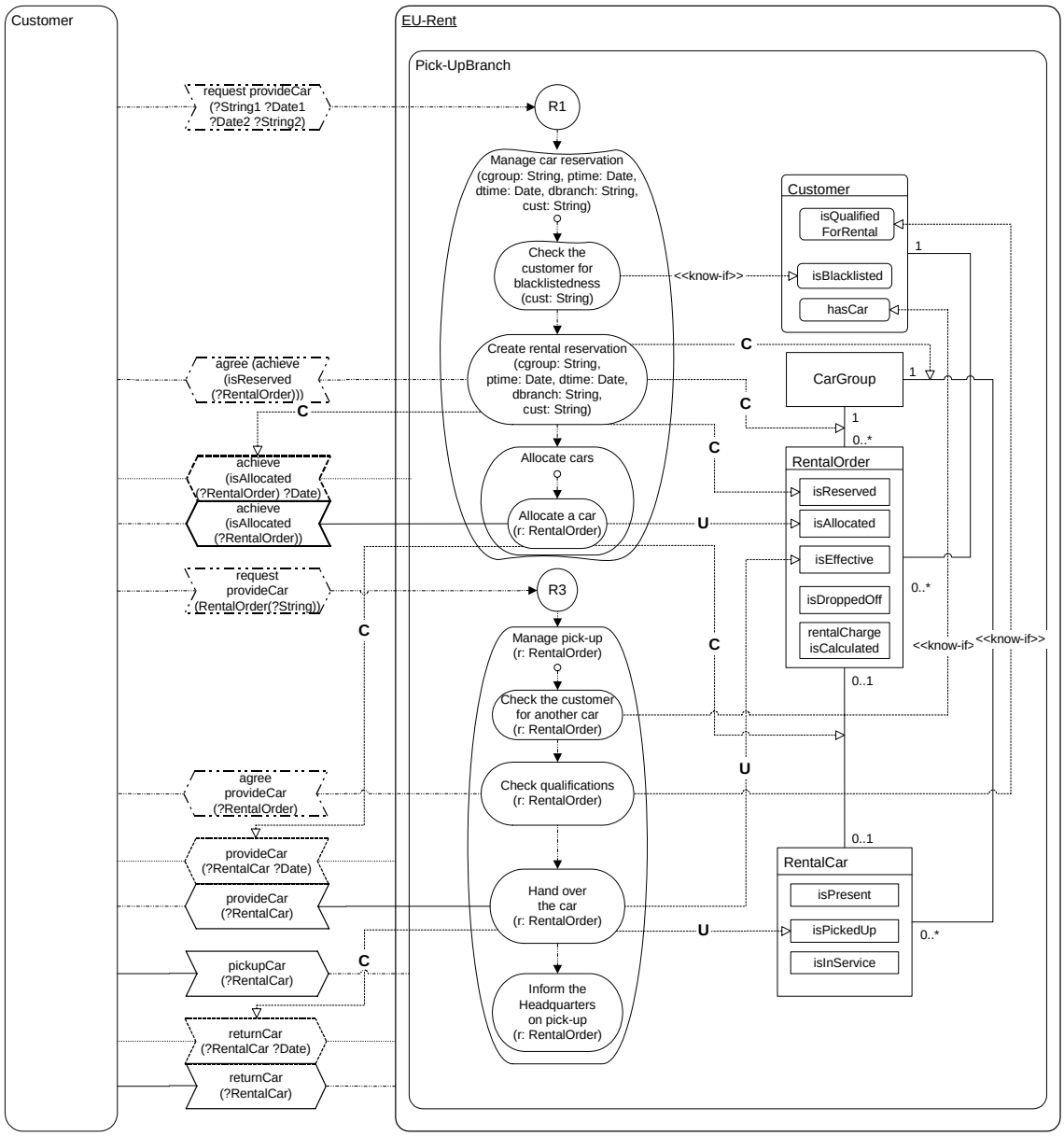


Figure 3-17. Activity diagram of the functional view for the business process type of car rental between the agent types Customer and Pick-Up Branch of the EU-Rent car rental company.

Please notice that while a use case of the type 'primary task' is always modelled from the perspective of its primary actor and the goal of the use case is thus the goal of the primary actor, in function modelling the perspective changes to that of the *agent (type) in focus* and the goal of the primary actor of the use case is *internalized* by the agent (type) in focus. For example, the use case "Pick up the car" initiated by the customer and the customer's goal to pick up the car reserved for him/her to be picked up at the pick-up branch at the pick-up time become the corresponding activity type "Manage car rental" of the institutional agent role Pick-Up Branch and the definition of its goal to provide the customer with the car.

The activity diagram in Figure 3-17 thus constitutes the function model of the business process types of reserving and picking up a rental car between the external agent type Customer and the focus agent type Pick-UpBranch of the EU-Rent car rental company. The activity diagram represents only the main success scenarios of these business process types.

3.8.4.2. Defining Preconditions and Goals

As we saw in section 3.6.7, *input parameters* can be defined for an activity type as formal parameters. At runtime, an activity instance can access through its instantiated input parameters the instances of entity and association types and action event and commitment/claim types making up the beliefs of the corresponding agent.

Preconditions and goals are defined for activity types as propositions by means of OCL. Table 3-18 presents the preconditions and goals defined for the activity types of the EU-Rent car rental company which are modelled in Figure 3-17.

At the step of function modelling, the precondition may be defined for an activity type in two cases: when an activity of the corresponding type is invoked by a reaction rule which is triggered by an external or internal agent or when the activity type defines a new input parameter. The first case is exemplified by the activity type "Manage pick-up" in Figure 3-17 and Table 3-18 because its instance is started by reaction rule R3 which is triggered by an external agent of the type Customer. As an example of the second case serves the activity type "Allocate a car" in Figure 3-17 which introduces the input parameter r of the object type RentalOrder referring to the instance of RentalOrder created within an activity of the type "Create rental reservation". The precondition of this activity type is therefore defined in Table 3-18 in terms of the attribute pickUpTime of the object type RentalOrder.

The goal, which is defined Table 3-18 for the activity type "Manage car reservation", was thoroughly explained section 3.6.2 in connection with the introduction of the mental effect categories.

The goal defined in Table 3-18 for the activity type "Check the customer for blacklistedness" specifies *knowing* by the Pick-UpBranch whether the customer in question is blacklisted or not. This goal is represented in OCL in terms of the status predicate isBlacklisted of the internal representation of the agent type Customer within the institutional agent type Pick-UpBranch.

The goal defined in Table 3-18 for the activity type "Create rental reservation" specifies that an instance of RentalOrder with the attribute values corresponding to the values of the activity's input parameters has been created, the relationships from it to the corresponding instances of CarGroup and Customer have been formed, and the *stit*-commitment to allocate a car to the rental order has been created.

The goal defined in Table 3-18 for the activity type "Allocate cars" specifies that cars have been allocated to all rental orders where a car is to be picked up on the following day.

The goal defined in Table 3-18 for the activity type "Allocate a car" specifies that a car has been allocated to the rental order, the corresponding *stit*-commitment has been deleted, and the *to-do*-commitment to provide the customer with the car by the pick-up time stated in the rental order has been created.

The goals defined in Table 3-18 for the activity types "Manage pick-up" and "Hand over the car" specify status changes of the corresponding instances of RentalOrder and RentalCar, the deletion of the *to-do*-commitment to provide the customer with the car, and the creation of the *to-do*-claim against the customer to return the car.

The goals defined in Table 3-18 for the activity types "Check the customer for another car" and "Check qualifications" are respectively specified as *knowing* whether the customer in question is blacklisted or not and whether the customer is qualified for renting a car or not. The goals are expressed in terms of the input parameter r of the type RentalOrder, referring to the given instance of RentalOrder.

Table 3-18. Source data items, preconditions, and goals pertaining to the activity types of Pick-UpBranch extracted at the step of function modelling.

Activity type and input parameter(s)	Precondition	Goal
Manage car reservation (cgroup : String, ptime : Date, dtime : Date, pbranch : String, dbranch : String, cust: String, senderID : String)	-	RentalOrder.allInstances->exists (r: RentalOrder | r.carGroupID = cgroup and r.pickUpTime = ptime and r.dropOffTime = dtime and r.pickUpBranchID = pbranch and r.dropOffBranchID = dbranch and r.carGroup->exists (cg : CarGroup | carGroupID = cgroup and cg->includes(r)) and r.customer->exists(c : Customer | personID = cust and c->includes(r)) and r.rentalCar->exists(c : RentalCar | c.carGroup = r.carGroup and c.rentalOrder = r) and r.isAllocated **and** provideCar.allInstances->exists (about = r.rentalCar and dueTime = ptime and sourceID = pbranch and targetID = cust))
Check the customer for blacklistedness (cust : String)	-	Customer.allInstances->any (personID = cust).isBlacklisted **or** not Customer.allInstances()->any (personID = cust).isBlacklisted
Create rental reservation (cgroup : String, ptime : Date, dtime : Date, pbranch: String, dbranch : String, cust : String)	-	RentalOrder.allInstances->exists (r: RentalOrder | r.carGroupID = cgroup and r.pickUpTime = ptime and r.dropOffTime = dtime and r.pickUpBranchID = pbranch and r.dropOffBranchID = dbranch and r.carGroup->exists (cg : CarGroup | carGroupID = cgroup and cg->includes(r)) and r.customer->exists(c : Customer | personID = cust and c->includes(r)) and r.isReserved and self.stitCommitmentClaim->exists (achieve = r.isAllocated and dueTime = ptime and sourceID = pbranch and targetID = senderID))
Allocate cars	-	RentalOrder.allInstances->select(isReserved and pickUpTime.date = now().date + 1)->forAll(isAllocated)
Allocate a car (r : RentalOrder)	RentalOrder. allInstances->exists (ro: RentalOrder | ro.isReserved and pickUpTime.date = now().date + 1 and ro = r)	r.rentalCar->exists (c : RentalCar | c.rentalOrder = r) **and** r.isAllocated **and** **not**(self.stitCommitmentClaim->exists (achieve = r.isAllocated and dueTime = r.pickUpTime and sourceID = r.pickUpBranchID and targetID = senderID) **and** provideCar.allInstances->exists (about = r.rentalCar and dueTime = r.pickUpTime and sourceID = r.pickUpBranchID and targetID = senderID))
Manage pick-up (r : RentalOrder, senderID : String)	RentalOrder. allInstances->exists (ro : RentalOrder | ro.isAllocated and ro = r)	r.isEffective and r.rentalCar.isPickedUp **and** **not**(provideCar.allInstances->exists (about = r.rentalCar and dueTime = r.pickUpTime and sourceID = r.pickUpBranchID and targetID = senderID)) **and** returnCar.allInstances->exists (about = r.rentalCar and dueTime = r.dropOffTime and sourceID = senderID and targetID = r.pickUpBranchID)
Check the customer for another car (r : RentalOrder)	-	r.customer.hasCar **or** **not** r.customer.hasCar
Check qualifications (r : RentalOrder)	-	r.customer.isQualifiedForRental **or** **not** r.customer.isQualifiedForRental

Table 3-18 (continued). Source data items, preconditions, and goals pertaining to the activity types of Pick-UpBranch extracted at the step of function modelling.

Hand over the car (r : RentalOrder)	-	r.isEffective and r.rentalCar.isPickedUp **and** **not**(provideCar.allInstances->exists (about = r.rentalCar and dueTime = r.pickUpTime and sourceID = r.pickUpBranchID and targetID = senderID)) **and** returnCar.allInstances->exists (about = r.rentalCar and dueTime = r.dropOffTime and sourceID = senderID and targetID = r.pickUpBranchID)
Inform the Headquarters on pick-up (r : RentalOrder)	-	-

3.8.5. Behaviour Modelling

While the functional view of agent-oriented modelling addresses the modelling of business functionality (what has to be done), the **behavioural view** addresses the modelling of business behaviour (in which order and under what conditions work has to be done). At the design step of function modelling, we extracted from goal-based use cases and described by means of activity diagrams unspecified activity types. At the step of behaviour modelling, we elaborate on these activity types by representing triggers and conditions for their performing and specifying completely as many activity types as possible. In accordance with Figure 3-10, the resulting *activity diagrams unite the models of the organizational, informational, interactional, functional, motivational, and behavioural views of agent-oriented modelling.*

3.8.5.1. Plans of Activity Types

In function modelling, an activity type is regarded as a "black box". This means that the composition of an activity type is not specified, and in principle any procedure or method which results, if it succeeds, in achieving the goal for an instance of the activity type can be applied as a *plan* of the activity type. In behaviour modelling, we explicitly specify a *plan* for an activity type. Generally, a ***plan*** is the means to achieve a goal [Presley97]. It should specify what is to be done (i.e. the goal to be achieved), the types of subactivities and actions included in the plan, and the constructs required for starting and sequencing them. A goal of an activity thus holds after successful execution of the plan that is defined for the activity type. Analogously to a goal, a plan also has an *assignee*, the agent or role type to which this plan has been assigned. A plan may recursively include activity types of the following three kinds:

- an ***automatic activity type*** with a complete created at the time of design or generated at runtime plan *P* for achieving a goal *G* where all subactivity types are completely specified in terms of automated actions;
- a ***human activity type*** with either just a goal *G* and no plan at all, or with a predefined plan *P* for achieving a goal *G* created at the time of design;
- a ***semiautomatic activity type*** with an incomplete created at the time of design or generated at runtime plan *P* for achieving a goal *G* where one or more subactivity types are human activity types.

Examples of automatic, human, and semiautomatic activity types in the case study of car rental are respectively "Allocate a car", "Hand over the car", and "Manage pick-up" of Pick-UpBranch. The latter consists of both automatic subactivity types (e.g., "Check the customer for another car") and a human subactivity type ("Hand over the car"). As we showed in section 3.6.6, even when an activity type does not have a plan, like certain human activity types, it is executable in an activity diagram.

The activity type and its subtypes are defined in Table 3-2 as the UML stereotypes Activity, AutomaticActivity, HumanActivity, and SemiautomaticActivity of the base class Class.

Within the Business Agents' Approach, plans are defined at the time of design by using activity diagrams. In this thesis, we thus do not treat generation of plans at runtime. The subject is discussed, for example, in [Wagner00b].

3.8.5.2. Complementing Activity Diagrams

In order to turn function models of business processes into behaviour models, goal-based use cases are re-examined with the intention to complement activity diagrams of the functional view according to the following four guidelines:

1. The <*time or sequence factor*> component of a scenario step is represented as an action or a non-action event type connected to the reaction rule that performs the action(s) described by the step in response to perceiving an event of the corresponding type.
2. The <*condition*> component of a scenario step is turned into the reaction rule that performs the action(s) described by the scenario step if the condition evaluates to true and the action(s) described by the corresponding step of the extension scenario, if any, in the opposite case.
3. The symbol for the type of action event to be performed or perceived or achieve construct to be made true within an activity of the given type as well as the arrow standing for a mental effect associated with an activity of the given type is connected to the reaction rule included by the activity type. Also an arrow denoting the activity starting action type is connected to the reaction rule.
4. When needed, a precondition or mental effect arrow of a reaction rule is augmented by an OCL expression as is described in sections 3.6.4 and 3.6.5.

According to guideline 1, the <*time or sequence factor*> component "*It is the end of the day*" of step 3 of the use case "Have a car reserved" presented in Table 3-4 is turned into reaction rule R1 depicted in Figure 3-19 that is invoked by the internal agent :Timer.

In compliance with guideline 2, the <*condition*> component of step 2 of the main scenario of the use case "Have a car reserved" in Table 3-4 is represented in Figure 3-18 as reaction rule R2 that performs the communicative action agree(achieve(isAllocated(?RentalOrder))), creates the rental order with the status isReserved, the associations between it and the Customer and CarGroup, and the *to-do* commitment to provide the customer with the car if the condition of the reaction rule evaluates to true. In the opposite case, reaction rule R2 performs the communicative action refuse provideCar(?String1 ?Date1 ?Date2 ?String2) and ends the business process of car rental with advance reservation. In the same way, the <*condition*> component of step 1 of the main scenario of the use case "Allocate a car" presented in Table 3-9 is turned into reaction rule R3 shown in Figure 3-19. This reaction rule allocates to the rental order a car of the car group requested and creates the corresponding *to-do* commitment to provide the customer with the car if the condition, which is expressed as the intensional predicate isAvailableOfOwnGroup(RentalOrder) attached to the object type RentalCar, returns true. In the opposite case, reaction rule R3 starts an activity of the type "Allocate a car of the next higher car group".

According to guideline 3, the symbols for the action event types pickupCar and provideCar in Figure 3-17, which are respectively perceived and performed by the Pick-UpBranch, are in Figure 3-18 connected to reaction rule R5 included by the activity type "Hand over the car". In addition, three mental effects that are in Figure 3-17 associated with this activity type are in Figure 3-18 connected to the symbol for reaction rule R5. Based on the same guideline, the arrow denoting in Figure 3-17 the activity starting action type START ACTIVITY "Hand over the car" is in Figure 3-18 connected to reaction rule R5. In addition to the activity type "Hand over the car", the elementary activity types "Create rental reservation" and "Check qualifications" in Figure 3-17 are specified in Figure 3-18 by means of reaction rules based on guideline 3. According to the same guideline, in Figure 3-19 reaction rules R3, R4, R7, R8, and R9 are connected to the *to-do* commitment/claim type provideCar(?RentalCar ?Date). However, only the first of such connections is shown in Figure 3-19 because of space limitations.

According to guideline 4, the precondition arrows of reaction rules R2, R3, R4, and R5, and the mental effect arrows of reaction rules R2, R4, and R5 in Figure 3-18 and of reaction rule R2 in Figure 3-19 are augmented by OCL expressions which are formed according to the principles stated in sections 3.6.4 and 3.6.5. The schema of a reaction rule may also require the introduction of new preconditions, like the precondition of reaction rule R5 in Figure 3-18 that is augmented by the OCL expression {rentalOrder = r}. The intent of this precondition is to complement the reaction rule's schema by the internal variable RentalCar.

The compositions of the activity types "Check the customer for blacklistedness", "Check the customer for another car", and "Inform the Headquarters on pick-up" in Figure 3-18 and "Allocate a car from another branch" in Figure 3-19 are presented in Appendix E.

For representing more complicated use case scenarios, like loops, the *behavioural patterns* described in section 3.8.5.3 are to be used. It is important to emphasize here that complementing

activity diagrams by behavioural constructs based on goal-based use cases is not as straightforward as transforming goal-based use cases into activity diagrams of the functional view. It is rather an *iterative process* where some behavioural constructs are "documented" by goal-based use cases only after they have been worked out in activity diagrams.

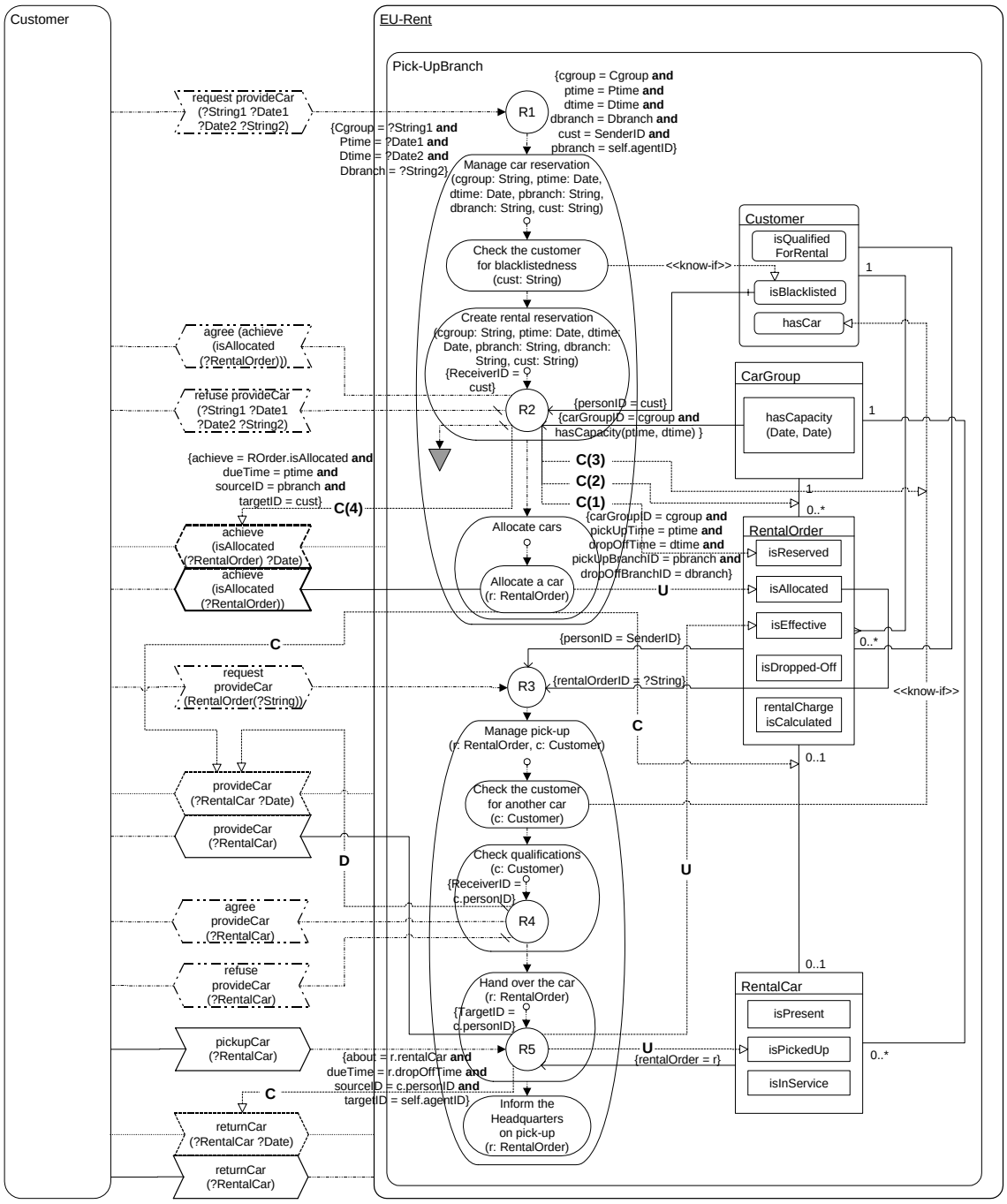


Figure 3-18. Activity diagram of the behavioural view for the business process type of car rental between the agent types Customer and Pick-UpBranch of the EU-Rent car rental company.

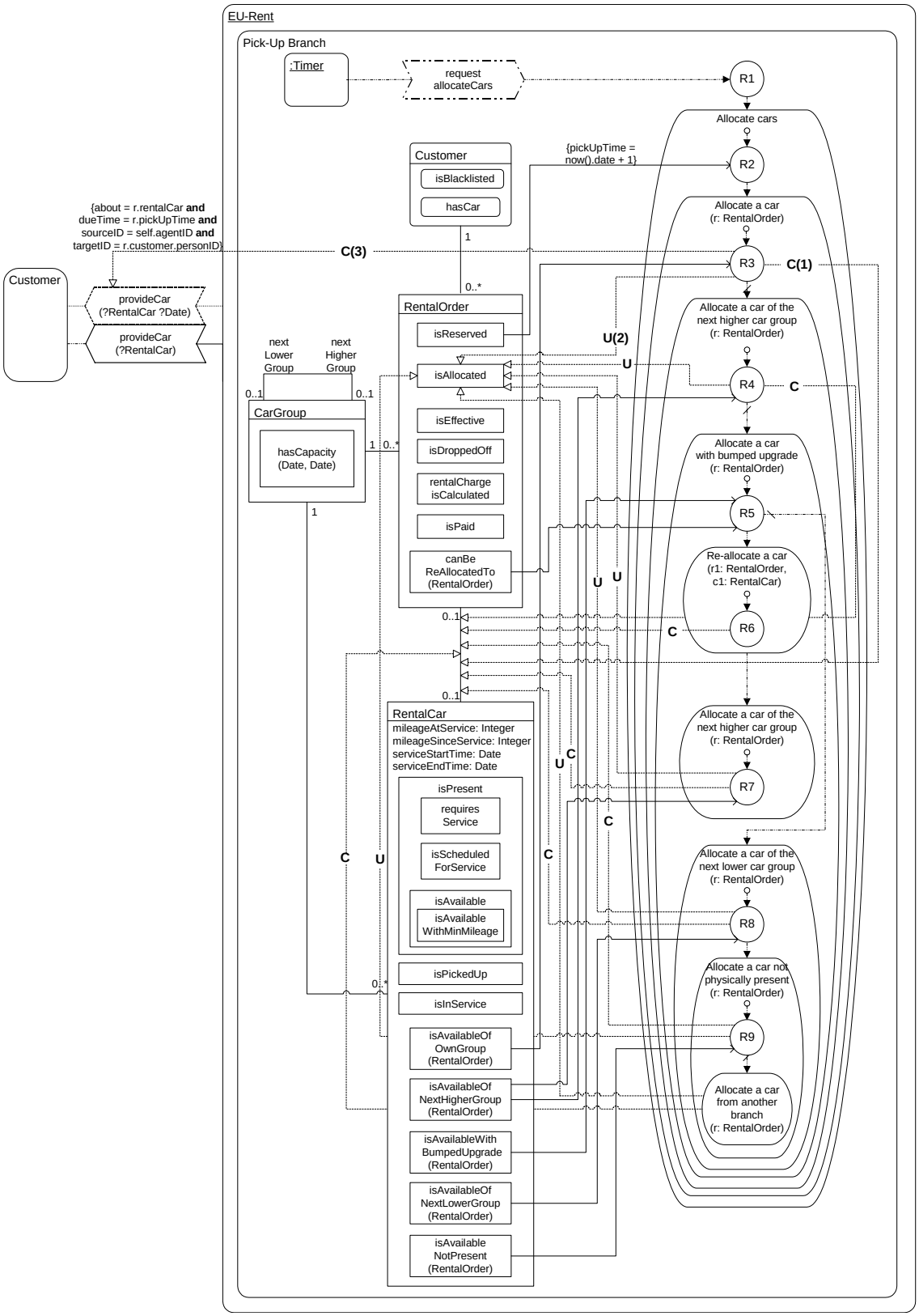


Figure 3-19. The composition of the activity type “Allocate a car”.

3.8.5.3. Behavioural Patterns

On the site [Patterns03] by Wil van der Aalst and others, 21 workflow patterns are distinguished. In addition to workflow management systems, these patterns can be used in business process modelling. These patterns are listed and shortly described in Table 3-19. In this section, we evaluate the support provided by the extended AORML for these patterns.

Table 3-19. Descriptions of workflow patterns.

Description of the pattern
Sequence - execute activities in sequence.
Parallel Split - execute activities in parallel.
Synchronization - synchronize two parallel threads of execution.
Exclusive Choice - choose one execution path from many alternatives.
Simple Merge - merge two alternative execution paths.
Multiple Choice - choose several execution paths from many alternatives.
Synchronizing Merge - merge many execution paths. Synchronize if many paths are taken. Simple merge if only one execution path is taken.
Multiple Merge - merge many execution paths without synchronizing.
Discriminator - merge many execution paths without synchronizing. Execute the subsequent activity only once.
N-out-of-M Join - merge many execution paths. Perform partial synchronization and execute subsequent activity only once.
Arbitrary Cycles - execute workflow graph without any structural restriction on loops.
Multiple Instances without Synchronization - generate many instances of one activity without synchronizing them afterwards
Multiple Instances with a Priori Known Design Time Knowledge - generate many instances of one activity when the number of instances is known at the design time.
Multiple Instances with a Priori Known Runtime Knowledge - generate many instances of one activity when the number of instances can be determined at some point during the runtime (as in parallel FOR loop).
Multiple Instances with no a Priori Runtime Knowledge - generate many instances of one activity when the number of instances cannot be determined beforehand (as in parallel WHILE loop).
Deferred Choice - execute one of several alternatives threads. The choice which thread is to be executed should be implicit.
Interleaved Parallel Routing – execute two activities in random order, but not in parallel.
Milestone - enable an activity until a milestone is reached.
Implicit Termination - terminate if there is nothing to be done.
Cancel Activity - cancel (disable) an enabled activity.
Cancel Case - cancel (disable) the process.

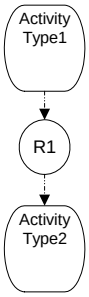


Figure 3-20. The pattern "Sequence".

The pattern "Sequence" ("Forced sequencing", "Sequential routing", "Serial routing") is represented in Figure 3-20. In the figure, an activity of the type ActivityType2 is started after the completion of an activity of the type ActivityType1. The circle symbol for the reaction rule is usually omitted from a graphical representation of this pattern. The example is expressed in the activity modelling language as follows:

```
ON END ActivityType1 THEN START ACTIVITY ActivityType2(...).
```

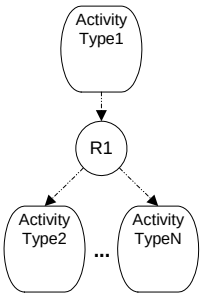


Figure 3-21. The pattern "Parallel Split".

The pattern "Parallel Split" ("AND-split", "Asynchronous spawning", "Parallel routing", "Fork") splits an activity into two or more activities which can be performed *in parallel*, thus allowing activities to be performed simultaneously or in any order. This pattern is exemplified by Figure 3-21. In the figure, after the end of an activity of the type ActivityType1, activities of the types ActivityType2 … ActivityTypeN are started to be performed in parallel. The example is expressed in the activity modelling language as follows:

ON END ActivityType1 THEN START ACTIVITY ActivityType2(...) … & START ACTIVITY ActivityTypeN(...).

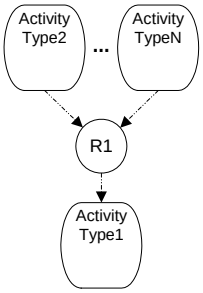


Figure 3-22. The pattern "Synchronization".

The pattern "Synchronization" ("AND-join", "Rendezvouz", "Synchronizer") merges two or more parallel activities into one activity. An example is presented in Figure 3-22. In the figure, after all parallel activities of the types ActivityType2 ... ActivityTypeN have ended, an activity of the type ActivityType1 is started. The example is expressed in the activity modelling language as follows:

ON END ActivityType2 ... AND END ActivityTypeN THEN START ACTIVITY ActivityType1(...).

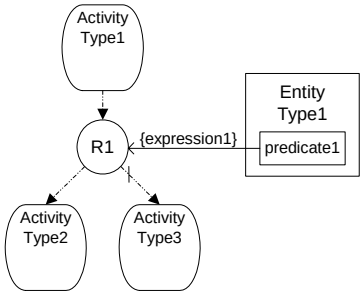


Figure 3-23. The pattern "Exclusive Choice".

The pattern "Exclusive Choice" ("XOR-split", "Conditional sequencing", "Conditional routing", "Switch", "Decision") chooses one of several activities for performing based on a control data. In the example of Figure 3-23, after the end of an activity of the type ActivityType1, if the condition specified by predicate1 and expression1 is true, an activity of the type ActivityType2 is started. Otherwise, an activity of the type ActivityType3 is started. The example is expressed in the activity modelling language as follows:

CONTEXT entity1 : EntityType1

ON END ActivityType1 IF EntityType1.allInstances->select(expression1 and predicate1)->includes(entity1) THEN START ACTIVITY ActivityType2(...) ELSE START ACTIVITY ActivityType3(...).

A construct of exclusive choice between more than two alternatives can be built by recursively inserting a next level exclusive choice into the activity type ActivityType3.

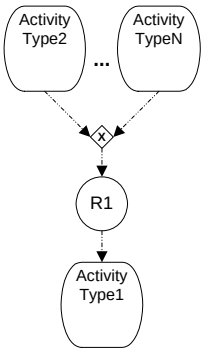


Figure 3-24. The pattern "Simple Merge".

The pattern "Simple Merge" ("XOR-join", "Asynchronous join", "Merge") starts an activity once any of the preceding *alternative* activities ends. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel. If this is not the case, the pattern "Multiple merge" or "Discriminator" should be applied instead. In the extended AORML, business process modelers are responsible for the model not having the possibility of parallel execution of alternative threads. In the example of Figure 3-24, an activity of the type ActivityType1 is started when *one* preceding activity out of alternative activities of the types ActivityType2 ... ActivityTypeN ends. An activity of the type ActivityType1 thus gets performed only once. The diamond symbol with the symbol 'X' inside stands for an exclusive disjunction (XOR). The example is expressed in the activity modelling language as follows:

ON END ActivityType2 … XOR END ActivityTypeN THEN START ACTIVITY ActivityType1(...).

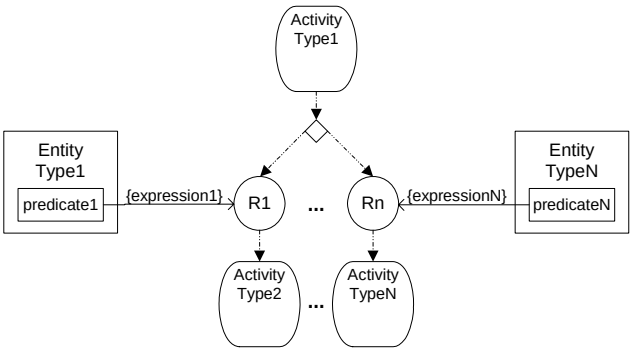


Figure 3-25. The pattern "Multiple Choice".

The pattern "Multiple Choice" ("OR-split", "Conditional routing", "Selection") is the generalization of the pattern "Exclusive Choice". It chooses *one or more* activities for performing based on a control data. In the example of Figure 3-25, according to reaction rule R1, after the end of an activity of the type ActivityType1, if the condition specified by predicate1 and expression1 is true, an activity of the type ActivityType2 is started. According to reaction rule Rn, if the condition specified by predicateN and expressionN is true, an activity of the type ActivityTypeN is started. Either an activity of one of the types ActivityType2 ... ActivityTypeN or any combination of them in parallel may thus get performed. An activity may get performed one or more times, depending on the condition specified by the given expression and predicate, after which the branch continues without synchronizing with the other branches. The example is expressed in the activity modelling language as follows:

CONTEXT entity1 : EntityType1, …, entityN : EntityTypeN
ON END ActivityType1 IF EntityType1.allInstances->select(expression1 and predicate1)->includes(entity1) THEN START ACTIVITY ActivityType2(...) ELSE …
…
ON END ActivityType1 IF EntityTypeN.allInstances->select(expressionN and predicateN)->includes(entityN) THEN START ACTIVITY ActivityTypeN(...) ELSE …

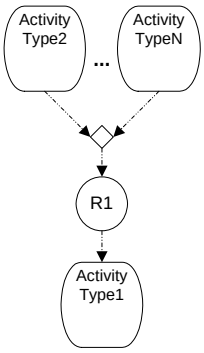


Figure 3-26. The pattern "Multiple Merge".

The pattern "Multiple Merge" ("OR-join") starts an activity *once for every preceding activity that ends*. In the example of Figure 3-26, an activity of the type ActivityType1 is started when *any* preceding activity out of *possibly parallel* activities of the types ActivityType2 ... ActivityTypeN ends. An activity of the type ActivityType1 thus gets performed as many times as the number of the preceding activities. The example is expressed in the activity modelling language as follows:

ON END ActivityType2 ... OR END ActivityTypeN THEN START ACTIVITY ActivityType1(...).

The pattern "Discriminator" models a point in a business process that waits for one of the preceding, *possibly parallel* activities to complete before starting the subsequent activity. From that moment on, it waits for all remaining preceding activities to complete and "ignores" them. Once all preceding activities have completed, it "resets" itself so that it can be started again. The pattern "Discriminator" naturally generalizes to the pattern "N-out-of-M-join" where N threads from M incoming transitions are synchronized. This generalized pattern "Discriminator" can be modelled as a reaction rule with a counter counting the number of the rule's triggering events of the type END ActivityType that have occurred. The example is expressed in the activity modelling language as follows:

ON END ActivityType2 ... OR END ActivityTypeN IF counter = count THEN START ACTIVITY ActivityType1(...) EFFECT counter = counter@pre + 1 ELSE EFFECT counter = counter@pre + 1.

The value of the variable counter is thus increased by one for every preceding activity that ends. The presupposition is that the value of counter is initially equal to 0 and the variable count contains the threshold number of the preceding activities. The initial values for counter and count can be given as the values of the input parameters of the activity enclosing the pattern.

According to [Patterns03], during the analysis/design time of a business process it is undesirable to be exposed to various syntactical constrains of a specific business process modelling tool such as for example that there should be only one entry and one exit point to the loop. The reason for this requirement is that most of the initial business process models contain arbitrary cycles at the analysis stage. Since activity diagrams do not impose any restrictions on the structure of cycles, a business process model in the Business Agents' Approach may have multiple entry and exit points which are represented as reaction rules. The Business Agents' Approach thus lends itself to the modelling of the "Arbitrary Cycles" pattern. Moreover, such arbitrary cycles are executable in our approach.

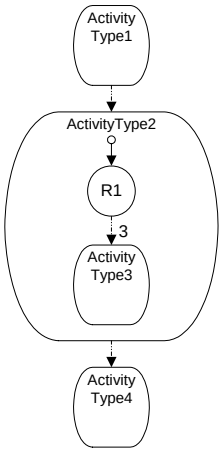


Figure 3-27. The pattern "Multiple Instances with a Priori Known Design Time Knowledge".

The pattern "Multiple Instances with a Priori Known Design Time Knowledge" enables to create many instances of one activity. The number of instances is known at the design time. In the example of the pattern in Figure 3-27, an activity of the type ActivityType1 is replicated three times to be executed in parallel. The example is expressed in the activity modelling language as follows:

ON END ActivityType1 THEN START ACTIVITY ActivityType2
ON START ActivityType2 THEN START ACTIVITY ActivityType3(...) 3 TIMES
ON END ActivityType2 THEN START ACTIVITY ActivityType3.

The activity type ActivityType2 acts as a *decomposition block*, using workflow terminology from [Aalst03a], whose task is to synchronize multiple instances of ActivityType3 so that when all subactivities of the type ActivityType3 have ended, the activity of the type ActivityType2 also ends, and the next activity of the type ActivityType4 is started (or the business process ends).

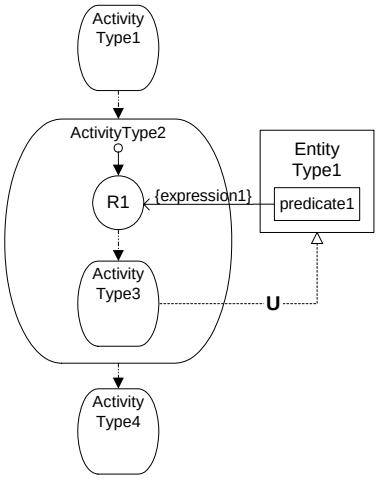


Figure 3-28. The pattern "Multiple Instances with a Priori Known Runtime Knowledge".

Within the pattern "Multiple Instances with a Priori Known Runtime Knowledge", the number of instances of a given activity type for a given business process type is variable and may depend on characteristics of the business process instance or availability of resources, but is known at some stage during runtime, before the instances of that activity type have to be created. Once all instances are completed, an activity of some other type needs to be started. In the Business Agents' Approach, this pattern is represented by its sub-pattern "Parallel For-Each" which is naturally supported by a reaction rule's mechanism of evaluating internal variables described in section 3.6.3. In Figure 3-28, the "Parallel For-Each" loop is included in an activity of the type ActivityType2 which is started after the end of an activity of the type ActivityType1. According to the example of the "Parallel For-Each" loop pattern represented in the figure, upon the start of an activity of the type ActivityType2, its subactivity of the type ActivityType3 is performed for each instance of the informational entity type (i.e., object type or a representation of an agent type) denoted by EntityType1 for which the precondition specified by expression1 and predicate1 evaluates to true. The activities of the type ActivityType3 are executed in parallel. The precondition may also be omitted. In that case, an activity of the type ActivityType3 is repeated for each instance of the informational entity type whose symbol the condition arrow is connected to. In the pattern depicted in Figure 3-28, implicit termination of an activity when all its subactivities have completed, as is described in section 3.6.6, is used as the synchronizing mechanism for multiple instances of ActivityType3. This is to say, when all subactivities of the type ActivityType3 have ended, the activity of the type ActivityType2 also ends, and the next activity of the type ActivityType4 is started (or the business process ends). The example represented in Figure 3-28 is expressed in the activity modelling language as follows:

CONTEXT entity1 : EntityType1
ON END ActivityType1 THEN START ACTIVITY ActivityType2
ON START ActivityType2 IF EntityType1.allInstances->select(expression1 and predicate1)->includes(entity1) THEN START ACTIVITY ActivityType3(...)
ON END ActivityType2 THEN START ACTIVITY ActivityType4.

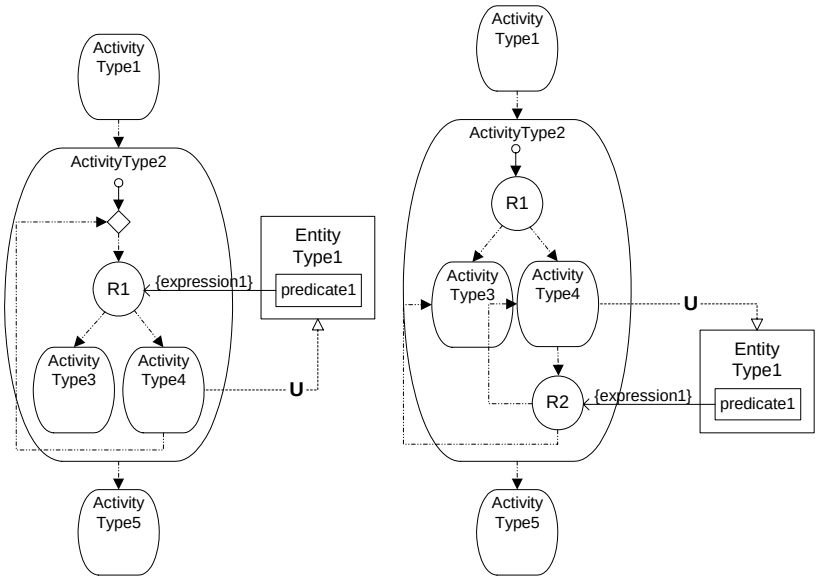


Figure 3-29. The patterns of "Multiple Instances with no a Priori Runtime Knowledge".

Within the pattern "Multiple Instances with no a Priori Runtime Knowledge", the number of instances of a given activity type is not known at the design time, nor it is known at any stage during runtime, until *immediately* before the instances of that activity type have to be created. Once all instances are completed, an activity of some other type needs to be started. The difference from the pattern "Multiple Instances with a Priori Runtime Knowledge" is that even while some of the activity instances are being executed or have already completed, new ones can be created. In the Business Agents' Approach, this pattern is represented by its sub-patterns "Parallel While-Repeat" and "Parallel Repeat-Until". In Figure 3-29, both loop patterns mentioned are included in an activity of the type ActivityType2 which is started after the end of an activity of the type ActivityType1. In these sub-patterns, an activity of the type ActivityType3 is invoked many times and as a rule an activity of the type ActivityType4 is used to determine if more instances of ActivityType3 are needed. An activity of the type ActivityType4 also updates an entity of the type EntityType1.

According to the example of the "Parallel While-Repeat" loop pattern in Figure 3-29 (on the left), upon the start of an activity of the type ActivityType2, if the condition specified by predicate1 and expression1 is true, reaction rule R1 starts activities of the types ActivityType3 and ActivityType4 to be executed in parallel. An activity of the type ActivityType4 checks if more instances of ActivityType3 are needed and records the decision in the corresponding entity of the type EntityType1. The completion of the activity of the type ActivityType3 again invokes reaction rule R1. If the condition specified by predicate1 and expression1 is true, reaction rule R1 creates a new parallel instance of ActivityType3 and a new instance of ActivityType4. This loop continues until all instances of ActivityType3 are completed.

Just like in case of the pattern "Multiple Instances with a Priori Runtime Knowledge", in both sub-patterns represented in Figure 3-29 synchronization of multiple instances of ActivityType3 is achieved through implicit termination of an activity when all its subactivities have completed, as is described in section 3.6.6. This means that when all subactivities of the type ActivityType3 have ended, the activity of the type ActivityType2 also ends, and the next activity of the type ActivityType5 is started (or the business process ends). The example of the sub-pattern "Parallel While-Repeat" represented in Figure 3-29 on the left is expressed in the activity modelling language as follows:

```
CONTEXT entity1 : EntityType1
ON END ActivityType1 THEN START ACTIVITY ActivityType2
ON START ActivityType2 OR END ActivityType4 IF EntityType1.allInstances->select(expression1 and predicate1)->includes(entity1) THEN START ACTIVITY ActivityType3(...) & START ACTIVITY ActivityType4(...)
ON END ActivityType2 THEN START ACTIVITY ActivityType5.
```

According to the example of the "Parallel While-Repeat" loop pattern in Figure 3-29 (on the right), upon the start of an activity of the type ActivityType2, reaction rule R1 starts activities of the types ActivityType3 and ActivityType4 to be executed in parallel. An activity of the type ActivityType4 checks if more instances of ActivityType3 are needed and records the decision in the corresponding entity of the type EntityType1. After the end of an activity of the type ActivityType4, if the condition specified by predicate1 and expression1 is true, reaction rule R2 creates a new parallel instance of ActivityType3 and a new instance of ActivityType4. This loop continues until all instances of ActivityType3 are completed. The example of the sub-pattern "Parallel Repeat-Until" is expressed in the activity modelling language as follows:

CONTEXT entity1 : EntityType1
ON END ActivityType1 THEN START ACTIVITY ActivityType2
ON START ActivityType2 THEN START ACTIVITY ActivityType3(...) & START ACTIVITY ActivityType4(...)
ON END ActivityType4 IF EntityType1.allInstances->select(expression1 and predicate1)->includes(entity1) THEN START ACTIVITY ActivityType3(...) & START ACTIVITY ActivityType4(...)
ON END ActivityType2 THEN START ACTIVITY ActivityType5.

In both sub-patterns described, the condition specified by predicate1 and expression1 determines at each step of the loop if more instances of the activity type ActivityType3 are needed. Since in many cases it is determined within the activity to be repeated whether more instances of it are needed (e.g., see the behavioural constructs modelled in section 4.1.5.5), both sub-patterns can be made sequential by merging the activity types ActivityType3 and ActivityType4.

The pattern "Deferred Choice" ("Dynamic XOR-split", "External choice", "Implicit choice") is a point in the business process where one of several activities is chosen to be performed. In contrast to the "XOR-Split" pattern, the choice is not made explicitly (e.g., based on a control data), but several alternatives are offered to the environment. The environment then selects the activity to be performed. Only one of the alternative activities is executed. This means that once the environment triggers one of the activities, the other alternative activities are withdrawn. In the Business Agents' Approach, a trigger by the environment is modelled as perceiving of an action event or a non-action event by the agent in focus.

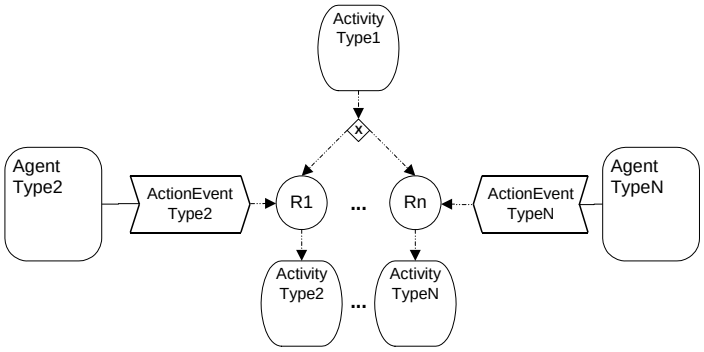


Figure 3-30. The pattern "Deferred Choice".

According to the example of the pattern "Deferred Choice" in Figure 3-30, after the end of an activity of the type ActivityType1, only one activity of the types ActivityType2 ... ActivityTypeN is started depending on which action event out of possible ones specified by ActionEventType2 ... ActionEventTypeN is *first perceived* by the agent. Notice the diamond symbol with the symbol 'X' inside standing for an exclusive disjunction (XOR). The example is expressed in the activity modelling language as follows where agentType2 ... agentTypeN stand for instances of the respective agent types:

ON END ActivityType1 THEN XOR (ON RECEIVE ActionEventType2 FROM agentType2 THEN START ACTIVITY ActivityType2(...) ... ON RECEIVE ActionEventTypeN FROM agentTypeN THEN START ACTIVITY ActivityTypeN(...)).

The activity modelling language defined in section 3.6.7 also enables to model the pattern "Deferred Choice" combined with an inclusive disjunction (OR). The meaning of such a pattern is that *one or several* activities out of activities of the types ActivityType2 ... ActivityTypeN is (are) started depending on which events are perceived by the agent.

The pattern "Implicit Termination" specifies that a given subprocess should be terminated when there are no activities being performed in the business process and no other activity can be started (and at the same time the process is not in deadlock). In our approach, this pattern is supported naturally through implicit termination of an activity when all its subactivities have completed (v. section 3.6.6).

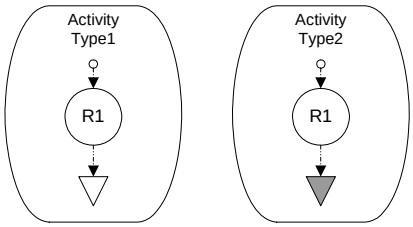


Figure 3-31. The patterns "Cancel Activity" and "Cancel Case".

The pattern "Cancel Activity" shown in Figure 3-31 (on the left) cancels the current activity being performed by removing the representation of the activity from the logical term describing the agent's activity state which was explained in section 3.6.6. It is not, however, possible to cancel other activity than the current one. The example is expressed in the activity modelling language as follows:

ON START ActivityType1 THEN CANCEL ActivityType1.

The pattern "Cancel Case" in Figure 3-31 (on the right) cancels the whole business process being executed by emptying the logical term describing the agent's activity state which was explained in section 3.6.6. The example is expressed in the activity modelling language as follows:

ON START ActivityType1 THEN CANCEL PROCESS.

According to the activity modelling language presented in section 3.6.7, reaction rules within behavioural constructs may have "side effects": in addition to starting an activity, a reaction rule within a behavioural construct like any other reaction rule may invoke actions or have mental effects. In many cases, this feature is capable of making activity diagrams shorter and more compact. For example, reaction rule R32 within the behavioural pattern "Deferred Choice" represented in Appendix F doesn't start any activity but invokes all the actions to be performed by itself.

Table 3-20, adopted from [Patterns03], summarizes the results of the comparison of various standards for workflow and business process modelling. We have complemented Table 3-20 with the column showing the support for the behavioural patterns by the extended AOR Modelling Language. In compliance with the notation used in [Patterns03], if the standard directly supports the pattern through one of its constructs, it is rated +. If the pattern is not directly supported but can be "mimicked", it is rated +/-. Any solution which results in spaghetti diagrams or coding is considered as giving no support and is rated -. Table 3-20 includes the following standards in addition to the extended AORML: XML Process Definition Language (XPDL) [XPDL], UML Activity Diagrams [OMG03a], BPEL4WS (Business Process Execution Language for Web Services) [BPEL], and BPML (Business Process Modelling Language) [BPML].

Table 3-20 reveals that the extended AORML supports all workflow patterns being compared with the exception of "Synchronizing Merge", "Interleaved Parallel Routing", and "Milestone". The extended AORML thus provides a stronger support for workflow patterns than any other standard listed in Table 3-20. The table also affirms the weakness of UML for business process modelling.

All the behavioural patterns described in this section can be specified as different combinations of *atomic* reaction rules of the form described in section 3.6.6, as is shown in Table 3-21. In addition, Table 3-21 contains definitions of the activity triggering patterns that are represented in Figure 3-8. The names of the action event types and informational entity types starting with small letters in Table 3-21 stand for instances of the respective types. Please note that since the extended AORML does not allow for the first-level parallel activities, the (parallel) subactivities in the behavioural patterns "Parallel Split", "Synchronization", "Multiple Choice", and "Deferred Choice" have an implicit superactivity of the type *Temp* in Table 3-21. However, according to the operational semantics of extended KPMC agents described in section 3.6.6, such an implicit superactivity is created only in case the behavioural pattern is not already included by some activity type.

Table 3-20. The comparison of workflow and business process modelling standards.

Pattern	**Standard**				
	XPDL	UML	BPEL	BPML	**AORML**
Sequence	+	+	+	+	+
Parallel Split	+	+	+	+	+
Synchronization	+	+	+	+	+
Exclusive Choice	+	+	+	+	+
Simple Merge	+	+	+	+	+
Multi Choice	+	-	+	-	+
Multi Merge	-	-	-	+/-	+
Discriminator	-	-	-	-	+/-
Synchronizing Merge	+	-	+	-	-
Arbitrary Cycles	+	-	-	-	+
MI with a Priori Design Time Knowledge	+	+	+	+	+
MI with a Priori Runtime Knowledge	-	+	-	-	+
MI without a Priori Runtime Knowledge	-	-	-	-	+
Deferred Choice	-	+	+	+	+
Interleaved Parallel Routing	-	-	+/-	-	-
Milestone	-	-	-	-	-
Implicit Termination	+	-	+	+	+
Cancel Activity	-	+	+	+	+
Cancel Case	-	+	+	+	+

Table 3-21. Combinations of reaction rules, corresponding to the behavioural patterns of business agents.

Behavioural pattern	**Reaction rule(s)**
Starting an activity by means of an action event (Fig. 3-8 a))	**start**(*ActivityType1*) ← **recvMsg**[*actionEventType1*, *agentType1*]
Starting a subactivity (Figure 3-8 b))	**start**$_{sub}$(*ActivityType2*) ← **start-of**(*ActivityType1*) **end**(*ActivityType1*) ← **end-of**(*ActivityType2*)[12]
Starting a subactivity upon perceiving an action event (Figure 3-8 c))	**start**$_{sub}$(*ActivityType2*) ← **start-of**(*ActivityType1*), **recvMsg**[*actionEventType1*, *agentType1*] **end**(*ActivityType1*) ← **end-of**(*ActivityType2*)[13]
Starting an activity by means of two (or more) events (Figure 3-8 d))	**start**(*ActivityType2*) ← **end-of**(*ActivityType1*), **recvMsg**[*actionEventType1*, *agentType1*]
Starting the next activity upon perceiving an action event (Figure 3-8 e))	**end**(*ActivityType1*), **start**(*ActivityType2*) ← **start-of**(*ActivityType1*), **recvMsg**[*actionEventType1*, *agentType1*]
Sequence (Figure 3-20)	**start**(*ActivityType2*) ← **end-of**(*ActivityType1*)
Parallel Split (Figure 3-21)	**start**(*Temp*) ← **end-of**(*ActivityType1*)[13] **start**$_{sub}$(*ActivityType2*), …, **start**$_{sub}$(*ActivityTypeN*) ← **start-of**(*Temp*)
Synchronization (Figure 3-22)	**end**(*Temp*) ← **end-of**(*ActivityType2*), …, **end-of**(*ActivityTypeN*)[13] **start**(*ActivityType1*) ← **end-of**(*Temp*)
Exclusive Choice (Figure 3-23)	**start**(*ActivityType2*) ← **end-of**(*ActivityType1*), *Cond* **start**(*ActivityType3*) ← **end-of**(*ActivityType1*), ¬*Cond*
Simple Merge (Figure 3-24)	**start**(*ActivityType1*) ← **end-of**(*ActivityType2*) … **start**(*ActivityType1*) ← **end-of**(*ActivityTypeN*)
Multiple Choice (Figure 3-25)	**start**(*Temp*) ← **end-of**(*ActivityType1*)[14] **start**$_{sub}$(*ActivityType2*) ← **start-of**(*Temp*), *Cond*$_1$ … **start**$_{sub}$(*ActivityTypeN*) ← **start-of**(*Temp*), *Cond*$_n$
Multiple Merge (Figure 3-26)	**start**(*ActivityType1*) ← **end-of**(*ActivityType2*) … **start**(*ActivityType1*) ← **end-of**(*ActivityTypeN*)
Discriminator	**start**(*ActivityType1*), *Eff* ← **end-of**(*ActivityType2*), *Cond* *Eff* ← **end-of**(*ActivityType2*), ¬*Cond* … **start**(*ActivityType1*), *Eff* ← **end-of**(*ActivityTypeN*), *Cond* *Eff* ← **end-of**(*ActivityTypeN*), ¬*Cond*
MI with a Priori Known Design Time Knowledge (Figure 3-27)	**start**(*ActivityType2*) ← **end-of**(*ActivityType1*) **start**$_{sub}$(*ActivityType3*), **start**$_{sub}$(*ActivityType3*), **start**$_{sub}$(*ActivityType3*) ← **start-of**(*ActivityType2*) **start**(*ActivityType4*) ← **end-of**(*ActivityType2*)
Parallel For-Each (Figure 3-28)	**start**(*ActivityType2*) ← **end-of**(*ActivityType1*) **start**$_{sub}$(*ActivityType3*) ← **start-of**(*ActivityType2*), *Cond* **start**(*ActivityType4*) ← **end-of**(*ActivityType2*)

[12] This rule is provided just for the sake of informativity: according to the execution model of a KPMC agent, an activity with subactivities ends when all its subactivities have ended.

[13] An implicit superactivity of the type *Temp* is created only in case the behavioural pattern is not already included by some activity type.

Table 3-21 (continued). Combinations of reaction rules, corresponding to the behavioural patterns of business agents.

Parallel While-Repeat (Figure 3-29)	**start**(*ActivityType2*) ← **end-of**(*ActivityType1*) **start**$_{sub}$(*ActivityType3*) ← **start-of**(*ActivityType2*), *Cond* **start**$_{sub}$(*ActivityType4*) ← **start-of**(*ActivityType2*), *Cond* **start**$_{sub}$(*ActivityType3*) ← **end-of**(*ActivityType4*), *Cond* **start**$_{sub}$(*ActivityType4*) ← **end-of**(*ActivityType4*), *Cond* **start**(*ActivityType5*) ← **end-of**(*ActivityType2*)
Parallel Repeat-Until (Figure 3-29)	**start**(*ActivityType2*) ← **end-of**(*ActivityType1*) **start**$_{sub}$(*ActivityType3*) ← **start-of**(*ActivityType2*) **start**$_{sub}$(*ActivityType4*) ← **start-of**(*ActivityType2*) **start**$_{sub}$(*ActivityType3*) ← **end-of**(*ActivityType4*), *Cond* **start**$_{sub}$(*ActivityType4*) ← **end-of**(*ActivityType4*), *Cond* **start**(*ActivityType5*) ← **end-of**(*ActivityType2*)
Deferred Choice (Figure 3-30)	**start**(*Temp*) ← **end-of**(*ActivityType1*)[14] **end**(*Temp*), **start**(*ActivityType2*) ← **recvMsg**[*actionEventType2*, *agentType2*], **start-of**(*Temp*) … **end**(*Temp*), **start**(*ActivityTypeN*) ← **recvMsg**[*actionEventTypeN*, *agentTypeN*], **start-of**(*Temp*)
Cancel Activity (Figure 3-31)	**remove**(*ActivityType1*) ← **start-of**(*ActivityType1*)
Cancel Case (Figure 3-31)	**remove-all** ← **start-of**(*ActivityType1*)

3.8.6. Mapping Activity Diagrams to the Constructs of JADE

The operational semantics of extended KPMC agents presented in section 3.6.6 can be mimicked on the JADE agent platform. This enables to perform simulations with the extended AORML models, like the ones created for the EU-Rent car rental company and for the case studies to be presented in Chapter 4. Mimicking operational semantics of KPMC agents on JADE complies with the principles of Model Driven Architecture (MDA) of OMG [MDA]. According to the overview of MDA provided in [KlasseObjecten], the MDA process defines three steps:

1. First, a model at a high level of abstraction that is independent of any implementation technology is built. This is called a Platform Independent Model (PIM). In the Business Agent's Approach, the models of the extended AORML serve as the PIM.
2. Next, the PIM is transformed into one or more Platform Specific Models (PSM). A PSM is tailored to specify the PIM in terms of the implementation constructs that are available in one specific implementation technology, which is JADE in our approach.
3. The final step is to transform a PSM to code. Because a PSM fits its technology very closely, this transformation is rather trivial. The complex step is the one in which a PIM is transformed to a PSM.

The JADE (Java Agent Development Environment) agent platform is described in [Bellifemine01] and [JADE]. It is a software framework to build agent systems in the Java programming language [JAVA] for the management of networked information resources in compliance with the FIPA specifications [FIPA] for interoperable intelligent multi-agent systems. In addition to providing an agent development model, JADE deals with all the aspects that are not peculiar to agent internals and that are independent of the applications, such as message transport, encoding and parsing, or agent life-cycle management. According to [Bellifemine01], JADE offers the following features to the agent programmer:

- FIPA-compliant distributed agent platform which can be split onto several hosts.
- Java Application Programmer's Interface to send/receive messages to/from other agents;
- Library of FIPA interaction protocols, such as Contract Net, ready to be used.
- Graphical user interface to manage several agents from the same Remote Management Agent.

According to [JADE], an agent must be able to carry out several concurrent tasks in response to different external events. In order to make agent management efficient, every JADE agent is composed of a single execution thread and all its tasks are modelled and can be implemented as instances of the Java object class jade.core.behaviours.Behaviour. The developer who wants to implement an agent-specific task should define one or more subclasses of Behaviour, instantiate them, and add the resulting behaviour objects to the agent task list. The jade.core.Agent class, which must be extended by agent programmers, exposes two methods: addBehaviour(Behaviour) and removeBehaviour (Behaviour), which allow management of the ready tasks' queue of a specific agent. By using them, behaviours and sub-behaviours can be added whenever needed. Adding a behaviour should be seen as a way to spawn a new (cooperative) execution thread within the agent. A scheduler, implemented by the base jade.core.Agent class and hidden to the programmer, carries out a round-robin non-preemptive scheduling policy among all behaviours available in the ready tasks' queue, executing a Behaviour-derived class until it will release control. Behaviours thus work just like co-operative threads, but there is no stack to be saved. Therefore, *the whole computation state must be maintained in instance variables of the Behaviour and its associated Agent*.

The abstract class jade.core.behaviours.Behaviour provides an abstract base class for modelling agent tasks, and it sets the basis for behaviour scheduling as it allows for state transitions (i.e. starting, blocking, and restarting a Java behaviour object). It has the predefined subclasses SimpleBehaviour and CompositeBehaviour. The first of them is further divided into the subclasses OneShotBehaviour and CyclicBehaviour, while the second one has the subclasses SequentialBehaviour and ParallelBehaviour. The functionality of a behaviour is included in its start() method. Another important method of a behaviour is the block() method which allows to block a behaviour object until some event happens (typically, until a message arrives). The jade.core.behaviours.Behaviour class also provides two placeholder methods, named onStart() and onEnd(). These methods can be overridden by user defined subclasses when some actions are to be executed before and after running behaviour execution. The functionality of a SequentialBehaviour and ParallelBehaviour is included in the method onStart() in place of action().

We will next treat by views of agent-oriented modelling how executable JADE-based models corresponding to executable models expressed by means of the extended AORML can be created.

3.8.6.1. Organizational and Informational View

When preparing extended AOR models for simulation on JADE, we first represent the types of institutional agents of the organization model, such as Customer, Branch, Headquarters, and AutomotiveServiceStation shown in Figure 3-11, as the corresponding subclasses of the JADE's class jade.core.Agent. The instances of these Java classes form the agents of the simulation environment. After that, we turn the object types and representations of agent types of the information model, like RentalOrder/Invoice, CarGroup, RentalCar, Proposal, and Customer represented within the agent type Branch in Figure 3-13, into the respective Java classes. Their instances form the VKB's of the corresponding agents. For example, instances of the object classes RentalOrder, CarGroup, RentalCar, Proposal, and Customer form the VKB of the corresponding agent instance of the agent class Branch. When there are several instances of such a class, they are represented as elements of a Java collection of the type HashMap, as is shown in the example below. The status and intensional predicates of an informational entity type within an agent's VKB are implemented as functions attached to the corresponding object class. Associations between informational entity types are represented as object references between instances of the corresponding object classes. For example, the agent type Branch and the object type RentalCar of the extended AORML are represented as the following Java classes:

```
public class Branch extends jade.core.Agent {

   /** Virtual Knowledge Base */
   private HashMap rentalOrder = new HashMap();
   private HashMap carGroup = new HashMap();
   private HashMap rentalCar = new HashMap();
   private HashMap proposal = new HashMap();
   private Customer customer;

   /** Information about ontology */
   private Codec codec = new SLCodec();
   private Ontology ontology = CarRentalOntology.getInstance();

   /** The ID of the branch */
   private String branchID;

   ...
}

public class RentalCar extends Car implements Concept {

   /** Attributes */
   private Date serviceStartTime;
   private Date serviceEndTime;
   private int mileageAtService;
   private int mileageSinceService;

   /** References */
   private Branch branch;
   private RentalOrder rentalOrder;

   /** Statuses */
   public static final boolean is_present = false;
   public static final boolean requires_service = false;
   public static final boolean is_scheduled_for_service = false;
   public static final boolean is_available = false;
   public static final boolean is_picked_up = false;
   public static final boolean is_in_service = false;

   /** Status predicates */
   public boolean isAvailable() {
      if ( is_present &&
           ! requires_service &&
           ! is_scheduled_for_service )
        return true;
      else
        return false;
   }
   ...
}
```

3.8.6.2. Interactional View

Agent messages are represented as instances of the JADE's object class jade.lang.acl.ACLMessage. In order to be able to interpret messages received from each other, JADE agents of the simulation environment must share a common knowledge of the structure of concepts, predicates, and actions included in agent messages. For that purpose, there is a JADE-based ontology that describes the concepts, predicates, and actions used in agent messages. Such an ontology corresponds to the union of shared object and action event types defined by the extended AORML models of the informational and interactional views. The ontology of a problem domain, like the ontology of the EU-Rent car rental company, extends the ontology defined in the JADE development library (jade.content.onto.Ontology). According to [JADE], the basic ontology of JADE enables to create application-specific ontologies describing the elements that agents can use within the messages exchanged by them. An ontology is characterized by one name, one (or more) basic ontology that it extends, and a set of element schemas. Element schemas are Java objects describing the structure of concepts, actions, and predicates that are allowed in agent messages. *Concepts*, which correspond to the shared object types like RentalOrder/Invoice and RentalCar appearing in the interaction model of the EU-Rent car rental company shown in Figure 3-16, are expressions that indicate entities with a complex structure that can be defined in terms of slots. Concepts typically make no sense if used directly as the content of an ACL message. In general they are referenced inside predicates and other concepts, like agent actions. *Primitives* are expressions that indicate atomic entities such as strings and integers. They correspond to data types defined in the informational view. A concept may include one or more primitives. *Agent actions*, which correspond to the non-communicative action event types like provideCar, pickupCar, and achieve defined by the interaction ontology of the EU-Rent car rental company, are special concepts that indicate actions that can be performed by some agents. Unlike "normal" concepts, they are meaningful contents of certain types of FIPA ACL messages such as messages of the type "request". *Predicates*, which correspond to the status predicates like isAllocated, hasCar, and isServiced used in the interaction model of the EU-Rent car rental company shown in Figure 3-16, are expressions that say something about the status of the world and can be true or false. An ontology for a given domain is thus a set of schemas defining the structure of the predicates, agent actions and concepts (basically their names and their slots) that are pertinent to that domain [JADE]. Each schema added to the ontology is associated with the corresponding Java class. For example, the schema for the RentalCar concept is associated with the object class RentalCar. When using the defined ontology, expressions indicating rental cars in agent messages are instances of the RentalCar class. In JADE, one must also define Java classes corresponding to agent actions, like provideCar, and predicates. All Java classes corresponding to concept schemas implement the Concept interface. Analogously, all Java classes corresponding to action and predicate schemas implement the AgentAction and Predicate interface, respectively. An excerpt from the ontology of the car rental company looks like as follows:

```
public class CarRentalOntology extends Ontology {

   // The name identifying this ontology
   public static final String ONTOLOGY_NAME = "Car-Rental-Ontology";

   // VOCABULARY
   public static final String RENTAL_ORDER = "RentalOrder";
   public static final String ORDER_CAR_GROUP_ID = "carGroupID";
   public static final String ORDER_PICK_UP_TIME = "pickUpTime";
   public static final String ORDER_DROP_OFF_TIME = "dropOffTime";
   public static final String ORDER_PICK_UP_BRANCH_ID = "pickUpBranchID";
   public static final String ORDER_DROP_OFF_BRANCH_ID = "dropOffBranchID";
   public static final String RENTAL_CAR = "RentalCar";

   ...
   public static final String PROVIDE_CAR = "provideCar";
   public static final String PROVIDE_CAR_CAR = "rentalCar";

   ...
   // Private constructor
   private CarRentalOntology() {
      // The car rental ontology extends the basic ontology
      super(ONTOLOGY_NAME, BasicOntology.getInstance());

   try {
      add(new ConceptSchema(RENTAL_ORDER), RentalOrder.class);
      add(new AgentActionSchema(PROVIDE_CAR), provideCar.class);
```

```
    ...
    // Structure of the schema for the RentalOrder concept
    ConceptSchema cs = (ConceptSchema) getSchema(RENTAL_ORDER);
    cs.add(ORDER_CAR_GROUP_ID,
                        (PrimitiveSchema)getSchema(BasicOntology.STRING));
    cs.add(ORDER_PICK_UP_TIME,
                        (PrimitiveSchema)getSchema(BasicOntology.DATE));

    ...
    // Structure of the schema for the provideCar agent action
    AgentActionSchema as1 = (AgentActionSchema) getSchema(PROVIDE_CAR);
    as1.add(PROVIDE_CAR_CAR, (ConceptSchema) getSchema(RENTAL_CAR));

    ...
  }
  catch (OntologyException oe) {
    oe.printStackTrace();
  }
```

3.8.6.3. Functional and Behavioural Views

In the JADE agent class corresponding to some institutional agent type modelled in the organizational view, we represent an activity type consisting of sequential subactivities, like the activity type "Manage car rental" represented in Figure 3-18, as the corresponding subclass of SequentialBehaviour. Analogously, an activity type with parallel subactivities, like the activity type "Allocate cars" modelled in Figure 3-19, is implemented as a subclass of ParallelBehaviour. Instances of SequentialBehaviour and ParallelBehaviour have one or more sub-behaviours. The difference between them is that a SequentialBehaviour executes its sub-behaviours in sequential order, while the "children" of a ParallelBehaviour are executed concurrently. Finally, an elementary activity type, like the activity type "Create rental reservation" in Figure 3-18, is represented as a subclass of OneShotBehaviour. An instance of OneShotBehaviour is executed only once.

The action() method of a OneShotBehaviour, containing the functionality of the behaviour, corresponds to the reaction rule included by the respective elementary activity type. The action() method of a OneShotBehaviour is invoked by the JADE agent platform when the behaviour is started. Accordingly, each instance of the subclass of OneShotBehaviour, corresponding to some elementary activity, has to be a "child" of an instance of SequentialBehaviour or ParallelBehaviour, corresponding to the "father" activity. A "child" behaviour is started by invoking the addSubBehaviour method of the "father" behaviour with the argument new ActivityType(...). For example, an instance of Create_rental_reservation in the example below is invoked by the corresponding instance of Manage_car_reservation which is a subclass of SequentialBehaviour. The outermost behaviour, like an instance of Manage_car_reservation in the example below, is directly a "child" of the agent which is added through calling of the addBehaviour method of the corresponding instance of jade.core.Agent with the argument new MainActivityType(...). Invocation of the onStart() or action() method of a behaviour straightforwardly corresponds to the occurrence of the *start-of-activity* activity border event of the corresponding activity. In a similar way, the method onEnd() of a behaviour instance is invoked upon ending the behaviour. Invocation of this method thus corresponds to the *end-of-activity* activity border event of the corresponding activity.

Essential parts of each agent of the simulation system are instances of the object classes MessageHandler and InputHandler. The class MessageHandler extends CyclicBehaviour of JADE. The instance of MessageHandler acts in cycles of waiting for an incoming ACL message, gets a message, if there is any, from the agent's event queue by using the receive() method of jade.core.Agent, analyzes it, and starts the appropriate business process by adding the behaviour corresponding to the outermost activity of the process to the agent with a call of the agent's method addBehaviour. The instance of InputHandler works like the MessageHandler but expects input from a human agent through a GUI.

The input parameters of an activity type are defined as the corresponding formal parameters of the constructor of the corresponding behaviour class. When needed, the input parameters are passed to the constructors of inner behaviour classes. For example, the constructor of the behaviour class Manage_car_ rental is invoked by the MessageHandler with the actual parameters corresponding to the behaviour's formal parameters cgroup, ptime, dtime, dbranch, and msg. These parameters are then stored into the instance attributes of Manage_car_reservation and are thereafter passed to its inner activities by invoking the corresponding addSubBehaviour methods with the arguments new ActivityType(...) like is shown in the following example:

```
class Manage_car_reservation extends SequentialBehaviour {
  /** Placeholder for the received message */
  private ACLMessage receivedMessage;
  /** The identifier of the car group */
  private String carGroupID;
  /** The pick-up time */
  private Date pickUpTime;
  /** The drop-off time */
  private Date dropOffTime;
  /** The identifier of the pick up branch */
  private String pickUpBranchID;
  /** The identifier of the drop off branch */
  private String dropOffBranchID;
  public Manage_car_reservation (String cgroup, Date ptime, Date dtime,
                          String pbranch, String dbranch, ACLMessage msg){
    super();
    carGroupID = cgroup;
    pickUpTime = ptime;
    dropOffTime = dtime;
    pickUpBranchID = pbranch;
    dropOffBranchID = dbranch;
    receivedMessage = msg;
  }
  public void onStart() {
    addSubBehaviour(new Check_the_customer_for_blacklistedness
      (receivedMessage));
    addSubBehaviour(new Create_rental_reservation(carGroupID, pickUpTime,
      dropOffTime, pickUpBranch, dropOffBranch, receivedMessage));
    addSubBehaviour(new Allocate_cars());
  }
}
```

Table 3-22 shows the mapping of the notions of the extended AORML to the constructs of the JADE agent development model.

Table 3-22. Mapping of notions of the extended AORML to the object classes and methods of JADE.

Notion of the extended AORML	Object class of JADE	Object method of JADE (if applicable)
Object type	Class	-
Agent type	jade.core.Agent	-
Elementary activity type	jade.core.behaviours. OneShotBehaviour	-
Sequential activity type	jade.core.behaviours. SequentialBehaviour	
Parallel activity type	jade.core.behaviours. ParallelBehaviour	
Execution cycle of a KPMC agent	jade.core.behaviours. CyclicBehaviour	-
Receiving a message from the agent's event queue	jade.core.Agent	public final ACLMessage receive()
Waiting for a message to be received	jade.core.behaviours. ReceiverBehaviour	public ReceiverBehaviour (Agent a, long millis, MessageTemplate mt)
Starting the first-level activity	jade.core.Agent	public void addBehaviour (Behaviour b)
Starting a subactivity	jade.core.behaviours. SequentialBehaviour	public void addSubBehaviour (Behaviour b)
Starting a parallel subactivity	jade.core.behaviours. ParallelBehaviour	public void addSubBehaviour (Behaviour b)
Start-of-activity activity border event	jade.core.behaviours. OneShotBehaviour	public abstract void action()
Start-of-activity activity border event	jade.core.behaviours. SequentialBehaviour, jade.core.behaviours. ParallelBehaviour	public abstract void onStart()
End-of-activity activity border event	jade.core.behaviours.Behaviour	public int onEnd()
Agent message	java.lang.acl.ACLMessage	-

4. CASE STUDIES

This chapter contains two full-fledged case studies where a ceramic factory and the domain of advertising are modelled by using the methodology that was introduced and explained in sections 3.7 and 3.8.

4.1. THE CASE STUDY OF A CERAMIC FACTORY

The modelling approach proposed in this thesis is first evaluated by using the case study of Tallinn Ceramic Factory Ltd. located in Tallinn, Estonia. By using the methodology proposed, we have created the information, organization, interaction, function, motivation, and behaviour models of the problem domain.

4.1.1. Overview of Tallinn Ceramic Factory Ltd.

Tallinn Ceramic Factory [TKT] was founded in 1934, when the brick factory was opened in Kopli, Tallinn. The production was divided all over Estonia at that time. Red clay bricks were very demanded in Estonia, Russia, and Finland. In addition to the bricks, the production of household and decorative ceramics was started in the 1950s. Local red clay from Joosu was used as the raw material. The factory was privatized in 1994 and bought by today's owners in 1999. The branches were privatized separately and Tallinn Ceramic Factory remained as one factory in Kopli. In 2000, the factory was named Tallinn Ceramic Factory Ltd. The raw material was changed – red clay was replaced with white stoneware. This enabled to concentrate mainly on the production of high quality tableware. The factory still produces decorative ceramics, different cooperation orders, and artists' sets.

More than half of the production is exported. During the last ten years, the biggest export partner has been Sweden. The best known clients of the factory are the Scandinavian biggest ceramic producers Boda Nova Höganäs Keramik AB, Guldkroken/Röda Bodan AB, Arabia, and Röstrand. The factory mainly produces product sets according to subcontractual orders for them. Different stove tiles for fireplaces, jugs, bowls, and handles for mugs are exported as raw or finished products. The factory has been represented with its own standard production in the biggest Scandinavian household goods' fair FORMEX in Sweden by its wholesalers for six years already. For several years, the factory has also taken part in the biggest European household goods fair in Frankfurt. Beside Sweden, the factory also exports to Finland, Japan, and Norway. The standard production exported consists mainly of different hand painted tableware sets. The factory also exports product sets produced according to special orders for restaurants and catalogues.

In the local market the production is sold in the factory shop in Kopli and in bigger storehouses. The factory also produces tableware sets for restaurants, pubs, and cafes, and business gifts and souvenirs by special orders. In the local market the factory mainly sells the hand painted tableware sets.

The factory uses local red clay from Joosu and stoneware imported from Germany as raw materials. All other raw materials come from Germany, the Czech Republic, and the United Kingdom. Red clay is used only for the production of decorative ceramics. White stoneware is the high temperature clay which must be burned at the temperature 1160 – 1200 Centigrade. The products are burned twice which gives them a better quality standard. As the factory accepts special orders for special designs, the number of restaurants and pubs among its clients is increasing. The best outfit of products is granted for a very long time as all the products are covered with transparent glaze and the raw material used is of the highest quality.

There are 65 people working in the factory. Most of the workers have 10 – 20 years of working experience and knowledge of ceramic production. The production process in the factory involves a lot of handwork. In fact, most of the production operations, with the exception of e.g. burning, are performed by hand at Tallinn Ceramic Factory. Also production schedules are created and updated manually.

4.1.2. Goals of the Case Study

Agent-oriented modelling of business/manufacturing processes of the problem domain of Tallinn Ceramic Factory is important for two reasons:

1. An agent-oriented modelling approach lends itself easily to simulation. The models of the problem domain worked out by following the methodology proposed by us can thus be quite straightforwardly turned into the implementation constructs of the actual simulation environment. We will briefly describe in section 4.1.5.6 how this was done for the case study of Tallinn Ceramic Factory. The simulation environment worked out also prepares for the forthcoming automation of the factory.
2. With the advent of virtual enterprises, a manufacturing enterprise should be capable of composing its business/manufacturing processes in a modular fashion so that if the factory receives at short notice a subcontractual order the satisfaction of which requires only a part of a full-length manufacturing process of the enterprise, the order would be scheduled and satisfied in a dynamic, flexible, and fast manner. One way to achieve this is to view a manufacturing enterprise as consisting of active entities – *agents* – so that each agent would be responsible for scheduling and performing manufacturing operations of a certain resource. This objective is already acute at Tallinn Ceramic Factory because a remarkable portion of the orders received by it are subcontractual orders for mug handles and stove tiles for fireplaces from Sweden.

According to [Tamm87], the so-called 'simulation modelling', related to the first reason above, is also one of the most practical means of conceptual analysis of a problem domain. Its main advantages are:

- learning of and experimenting on the target system with complex internal dependencies;
- trying out the influences of decisions of informational, technological, and organizational nature;
- full understanding of the problem domain;
- enabling the selection of the most crucial objects, relationships between them, and rules needed for the creation of the conceptual model of the problem domain;
- connection to new situations that have not been seen in practice yet;
- flexibility with respect to operational time;
- discovering and solving problems related to the existing information/manufacturing systems.

With an agent-oriented approach to modelling and simulation of production environments, each agent is autonomous and does not know the decision logic of the other agents, as a rule. The decision logic is thus specified for each agent individually and not for the system of agents, as a whole. This is closer to how the real world "works" than traditional approaches for modelling big systems, including UML [OMG03a].

The second reason above is in line with the four key requirements for manufacturing control systems that are vital in practice identified in [Burmeister98]:

- the control system should be able to dynamically incorporate incoming orders;
- it should adapt to disturbances concerning orders and resources;
- it should exhibit more flexibility when re-arranging (the control of) the production process – ideally perform the reorganization itself;
- it should be able to co-ordinate its actions with other control systems.

Especially the first and second requirements listed are relevant for the current case study. The importance of subcontracting is stressed, e.g., in [Zeng99]: "In manufacturing, managers face 'make or buy' decisions, i.e., the choice of making components/products in house or subcontracting them to outside sources … These decisions are critical in today's highly competitive and dynamic business environment".

4.1.3. Principles of Reactive Scheduling

According to [Smith90], the job-shop scheduling problem (or factory scheduling problem) can be defined as one of coordinating sequences of manufacturing operations for multiple orders so as to:

- obey the temporal restrictions of production processes and the capacity limitations of a set of shared resources (e.g., machines), and
- achieve a satisfactory compromise with respect to a myriad of conflicting preferential constraints (e.g. meeting due dates, minimizing work-in-progress, etc.).

Two kinds of scheduling are distinguished between in [Smith90]:

- *predictive scheduling* which concerns an ability to effectively predict shop behaviour through the generation of production plans that reflect both the full complexity of the factory environment and the stated objectives of the organization;
- *reactive scheduling* which concerns an ability to intelligently react to changing circumstances, as the shop floor is a dynamic environment where unexpected events (e.g., machine breakdowns, quality control inspection failures) continually force changes to planned activities.

In the paper [Smith90], the OPIS (OPportunistic Intelligent Scheduler) factory scheduling system based on a common view of predictive and reactive scheduling is described. The OPIS scheduling architecture is derived from principles of standard blackboard style architectures. It assumes an organization comprised of a number of *knowledge sources* that extend, revise and analyze the globally accessible factory schedule. The OPIS scheduling architecture combines two principal components: a schedule maintenance subsystem, for incrementally maintaining a representation of current solution constraints, and an event-driven control cycle for coordinating the use of knowledge sources. Generally, coordination of the scheduling effort by the OPIS scheduler proceeds as an event-driven process. Changes in the state of the schedule, introduced either by internal problem-solving activity (e.g., generating a schedule for a given order) or by external factory status updates (e.g., notification of a machine breakdown) are detected by the schedule maintenance system and posted as control events to the system at the beginning of each problem solving cycle [Smith90].

The OPIS scheduling system combines two alternative problem solving perspectives in generating a schedule. An ***order-based perspective*** repeatedly focuses the scheduler on an individual order's schedule and promotes achievement of good compromises with respect to conflicts involving the operations that must be performed to produce a given order. Alternatively, a ***resource-based perspective*** isolates a specific resource schedule as the scheduler's focus of attention, and emphasizes resolution of conflicts involving operations that must compete for that resource. In the OPIS scheduler these perspectives are represented by the Order Scheduler and Resource Scheduler knowledge sources, respectively. The *Order Scheduler* provides a method for generating or revising scheduling decisions relative to some contiguous portion of a specific order's production plan. Its scheduling method uses a beam search to explore alternative sets of resource assignments and execution intervals with respect to relevant preference constraints (e.g. work-in-process time objectives, machine preferences, etc.). The *Resource Scheduler* provides a method for generating or revising the schedule of a designated resource. It generates scheduling decisions using an iterative dispatch-based approach, adding another operation to the schedule of the resource under consideration at each cycle, and emphasizes efficient resource utilization [Smith90].

Similarly, in the agent-based cooperative scheduling system described in [Ow88], the order- and resource-based perspectives are represented by the work-order manager and resource broker, respectively. The *work-order manager* is an agent whose role is to provide estimates of completion dates for prospective work-orders that minimize the completion time and work-in-process time of the work-orders. In addition, when a work-order is accepted, the work-order manager is responsible for finalizing the contracts with the resource brokers, thereby causing the order to be scheduled. The work-order manager has access to information about the operations that the various resource brokers can perform, and the manufacturing requirements of prospective work-orders. A *resource broker* is an agent representing a set of resources which can perform similar operations. Resource brokers can represent machines, storage areas, tools, skill levels etc. When a resource broker receives a call from the work-order manager, bids are generated in accordance with the following hard constraints:

- resources/machines must be able to perform the operation, e.g. fulfill machining requirements;
- the maximum capacity of the resource should not be exceeded, e.g., each machine can only work on one work-order at a time;
- precedence constraints – i.e., an operation cannot start until all its preceding operations have completed.

In reactive scheduling, violation of the last two constraints resulting from constraint propagation in response to schedule changes can lead to the detection of two types of conflicts [Smith95]:

- capacity conflicts - situations where the resource requirements of a set of currently scheduled operations exceed the available capacity of a specific resource over some interval of time;
- time conflicts - situations where either the time bounds or scheduled execution times of two operations belonging to the same process instantiation violate a defined temporal precedence constraint.

The recognition of such conflicts signals the need for schedule revision. Detected conflicts are posted in the current control state as *elementary conflict events* which require subsequent scheduling attention [Smith95].

Constraint propagation can also lead to detection of rescheduling opportunities, situations where time and capacity constraints are loosened by introduced schedule changes. In the current implementation of OPIS, such situations are treated in a somewhat specialized manner; opportunity events are posted only in response to changes originating from external events that imply additional resource capacity (e.g., cancellation of a process request) to ensure that a rescheduling process is triggered [Smith95].

According to [Smith95], the simplest reactive methods invoked in response to conflict and opportunity events in OPIS are the Right Shifter and Left Shifter, respectively.

The ***Right Shifter*** implements a reactive method that resolves conflicts by simply "pushing" the scheduled execution times of designated operations forward in time. Execution of these designated shifts can introduce both time conflicts (with downstream operations belonging to the same process) and capacity conflicts (with operations scheduled downstream on the same resource). However, these conflicts are internally resolved by recursively propagating the shifts through resource and process schedules to the extent necessary. Thus, the Right Shifter will not introduce any new conflicts into the overall schedule.

The ***Left Shifter*** provides a similar but totally non-disruptive reactive method that "pulls" operations backwards in time (i.e., closer to execution) to the extent that current resource availability and temporal process constraints will permit. The method proceeds by sliding operations on a designated resource *R* to exploit an identified interval of available resource capacity (and any capacity intervals created by this sliding), and then recursively applying the procedure to the resources associated with the successor operations of processes who have had their scheduled execution interval on *R* changed. The recursion terminates whenever a downstream resource schedule is encountered that does not provide opportunities for left shifting or when process schedules have been completely traversed [Smith95].

The scheduling system of the ceramic factory to be modelled and simulated in the present thesis will be based on a mixture of the OPIS system [Smith90] and the agent-based cooperative scheduling system [Ow88] described above. The *production department* and a *resource unit* of the factory respectively embody the work-order manager and a resource broker. Since with an agent-oriented approach there is no centralized representation of the schedule like in OPIS, the production department has an overview of the schedule from the perspective of a production order and its production activities, while a resource unit knows about utilization of the resources represented by it. The Right Shifter and Left Shifter are realized through rescheduling the production activity or activities whose schedules are to be shifted right or left.

4.1.4. Analysis with Goal-Based Use Cases

The use cases in Tables 4-1 – 4-26 describe the business/manufacturing process types of the ceramic factory with different types of internal actors of the factory, sketched as a part of the analysis step, in focus. In Table 4-1, use case 1 "Have the product set produced", which has the sales department of the ceramic factory in focus, is presented. This use case is triggered by receiving from a customer a request to have the product set specified by the product code and required quantity produced. The goal of the use case, "expecting the product set to be produced" is given in its context in an informal way. It is semi-formalized in section 4.1.5.4 at the modelling phase of design. The use case is modelled from the perspective of the customer with the sales department in focus (*scope*) which means that the goal of the use case is the so-called *user goal*, the goal of the actor (i.e., the customer) trying to get work (*primary task*) done. Since the use case "Have the product set produced" is triggered by the customer, the customer is called the *primary actor* of this use case. The production department is termed the *secondary actor* because it is the one from which the actor in focus, sales department, needs assistance to satisfy the user goal internalized by it. The production department, in turn, has resource units as secondary actors. Other primary tasks, i.e. use cases that are triggered by primary actors, are use cases 5, 6, 7, 11, 15, 17, 19 and 21 below.

Use case 1 includes as *subfunctions* use cases 2, 3, and 4. As we learned in section 3.7.1, the goal of a *subfunction*, which is a subgoal of some user goal, is attached to the actor in focus. For example, the goal "expecting the production order to be scheduled" of the subfunction "Have the production order scheduled" (use case 2), which is a subgoal of the user goal "expecting the product set to be produced", is attached to the sales department. Among the use cases mentioned, use case 3 "Make a proposal and process the reply" includes the main scenario for the case the production proposal is accepted by the customer and the *extension scenario* for the opposite case.

A special group of subfunctions are subfunctions that are triggered by internal actors. In the example of the ceramic factory, to this group belong use cases 22, 23, and 24 below which are triggered by an internal worker.

In Tables 4-1 to 4-26, the <*time or sequence factor*> and <*condition*> components of use case steps are distinguished by representing them in *italic*.

Table 4-1. Extended use case for the business process "Have the product set produced".

USE CASE 1	Have the product set produced.	
Goal in Context	The customer expects the product set to be produced.	
Scope & Level	Sales department, primary task.	
Preconditions		
Success End Condition	The product set has been produced.	
Primary Actor **Secondary Actors**	Customer. Production department.	
Trigger	A request by the customer to provide it with a product set, specifying the product code and the quantity required.	
DESCRIPTION	**Step**	**Action**
	1	The sales department creates the production order.
	2	The sales department has the production order scheduled (**Use Case 2**).
	3	The sales department makes a proposal to the customer and processes the reply by the customer (**Use Case 3**).
	4	The sales department has the production order completed (**Use Case 4**).

Table 4-2. Extended use case for the business process "Have the production order scheduled".

USE CASE 2	Have the production order scheduled.	
Goal in Context	The sales department expects the production order to be scheduled.	
Scope & Level	Sales department, subfunction.	
Preconditions	The sales department has created the production order.	
Success End Condition	The production order has been scheduled.	
Primary Actor **Secondary Actors**	Customer. Production department.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The sales department forwards the production order to the production department for scheduling.
	2	The sales department receives the production order from the production department and registers the scheduling.

Table 4-3. Extended use case for the business process "Make a proposal and process the reply".

USE CASE 3	Make a proposal and process the reply.	
Goal in Context	The sales department expects the production proposal to be accepted or rejected by the customer.	
Scope & Level	Sales department, subfunction.	
Preconditions	The production order has been scheduled.	
Success End Condition	The production proposal has been accepted or rejected.	
Primary Actor **Secondary Actors**	Customer. Production department.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	A proposal to the customer is authorized by the sales manager, registered, and sent to the customer.
	2	*The acceptance of the proposal is received from the customer*: the acceptance is registered and the sales department commits to provide the customer with the product set, corresponding to the production order.
EXTENSIONS	**Step**	**Branching Action**
	2a	*The rejection of the proposal is received from the customer*: the rejection is registered, the production department is requested to delete the production order, and the business process ends.

Table 4-4. Extended use case for the business process "Have the production order completed".

USE CASE 4	Have the production order completed.	
Goal in Context	The sales department expects the product set to be produced according to the given production order.	
Scope & Level	Sales department, subfunction.	
Preconditions	The production proposal has been accepted by the customer.	
Success End Condition	The product set has been produced.	
Primary Actor **Secondary Actors**	Customer. Production department.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The sales department requests the production department to complete the production order
	2	The sales department receives a notification about the completion of the production order from the production department and registers it.
	3	The sales department informs the customer about the completion of the production order.

Table 4-5. Extended use case for the business process "Have the product set delivered".

USE CASE 5	Have the product set delivered.	
Goal in Context	The customer expects the product set to be delivered.	
Scope & Level	Completed production store (internal actor of the sales department), primary task.	
Preconditions	The product set has been produced.	
Success End Condition	The product set has been delivered to the customer.	
Primary Actor Secondary Actors	Customer.	
Trigger	A request by the customer to release the product set.	
DESCRIPTION	**Step**	**Action**
	1	The completed production store delivers the product set to the customer, discharges the commitment to provide the customer with the product set, and registers the delivery.
	2	The completed production store creates the invoice.
	3	*The invoice is authorized by the sales manager*: the invoice is sent to the customer, and the sending is registered.
	4	The completed production store creates a claim against the customer to pay for the product set according to the invoice.

Table 4-6. Extended use case for the business process "Register the payment".

USE CASE 6	Register the payment.	
Goal in Context	The customer wants to pay for the product set delivered according to the invoice.	
Scope & Level	Sales department, primary task.	
Preconditions	The product set has been delivered and the invoice has been sent to the customer.	
Success End Condition	The customer has paid for the product set delivered according to the invoice.	
Primary Actor Secondary Actors	Customer.	
Trigger	Receiving of a payment by the customer.	
DESCRIPTION	**Step**	**Action**
	1	The sales department registers the payment and satisfies the claim against the customer to pay for the product set according to the invoice.

Table 4-7. Extended use case for the business process "Create the product set and schedule the production order".

USE CASE 7	Create the product set and schedule the production order.	
Goal in Context	The sales department expects the product set to be created and the production order to be scheduled.	
Scope & Level	Production department, primary task.	
Preconditions		
Success End Condition	The product set has been created and the production order has been scheduled.	
Primary Actor Secondary Actors	Sales department. Resource units.	
Trigger	A request by the sales department to schedule the production order.	
DESCRIPTION	**Step**	**Action**
	1	The production department creates the instance of the product set, corresponding to the production order.
	2	The production department instantiates the production plan (**Use Case 8**).
	3	The production department schedules the production order (**Use Case 9**).
	4	The production department sends the production order to the sales department

Table 4-8. Extended use case for the business process "Instantiate the production plan".

USE CASE 8	Instantiate the production plan.	
Goal in Context	The production department expects the production plan corresponding to the production order to be instantiated.	
Scope & Level	Production department, subfunction.	
Preconditions	The production department has created the instance of the product set, corresponding to the production order.	
Success End Condition	The production plan for the product set has been instantiated.	
Primary Actor Secondary Actors	Sales department.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each production activity type of the product type of the product set*: create instance of the production activity type.

Table 4-9. Extended use case for the business process "Schedule the production order".

USE CASE 9	Schedule the production order.	
Goal in Context	The production department expects the production activities of the production plan to be scheduled.	
Scope & Level	Production department, subfunction.	
Preconditions	The production plan corresponding to the production order has been instantiated.	
Success End Condition	The production order has been scheduled.	
Primary Actor Secondary Actors	Sales department. Resource units.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each production activity of the production plan in the order of their performing*: have the production activity scheduled (**Use Case 10**).

Table 4-10. Extended use case for the business process "Have the production activity scheduled".

USE CASE 10	Have the production activity scheduled.	
Goal in Context	The production department expects the production activity to be scheduled.	
Scope & Level	Production department, subfunction.	
Preconditions	The production activity has been instantiated as a part of the corresponding production plan.	
Success End Condition	The production activity has been scheduled.	
Primary Actor Secondary Actors	Sales department. Resource units.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The production department sets the earliest start time of the production activity and sends to the corresponding resource unit a request to schedule the production activity.
	2	The production department receives from the resource unit a confirmation of the scheduling of the production activity and registers the scheduling.

Table 4-11. Extended use case for the business process "Complete the production order".

USE CASE 11	Complete the production order.	
Goal in Context	The sales department expects the production order to be completed.	
Scope & Level	Production department, primary task.	
Preconditions	The production order has been scheduled.	
Success End Condition	The production order has been completed.	
Primary Actor **Secondary Actors**	Sales department. Resource units.	
Trigger	A request to complete the production order by the sales department.	
DESCRIPTION	**Step**	**Action**
	1	The production department commits towards the sales department to complete the production order.
	2	*Until the production order is completed*: the production department follows the production activities of the production plan corresponding to the production order for rescheduling, start, and completion (**Use Case 12**).
	3	The production department informs the sales department about the completion of the production order and discharges the commitment towards the sales department to complete the production order.

Table 4-12. Extended use case for the business process "Follow the production activities".

USE CASE 12	Follow the production activities.	
Goal in Context	The production department expects the status changes of the production activities of the given product set to be registered.	
Scope & Level	Production department, subfunction.	
Preconditions	All the production activities of the given product set have been scheduled.	
Success End Condition	All the production activities of the given product set have been completed.	
Primary Actor **Secondary Actors**	Sales department. Resource units.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*The production department receives from a resource unit a message about rescheduling of a production activity involved in a capacity conflict*: the production department has all the production activities to be performed after the given production activity that are involved in a time conflict shifted (**Use Case 13**).
EXTENSIONS	**Step**	**Branching Action**
	1a	*The production department receives from a resource unit a message about the start of a production activity*: the start of the production activity is registered.
	1b	*The production department receives from a resource unit a message about the end of a production activity*: the end of the production activity is registered and the production department has all the production activities to be performed after the given production activity that are involved in a time conflict shifted (**Use Case 13**).

Table 4-13. Extended use case for the business process "Have the production activities shifted".

USE CASE 13	Have the production activities shifted.	
Goal in Context	The production department expects all the production activities to be performed after the given production activity that are involved in a time conflict to be shifted.	
Scope & Level	Production department, subfunction.	
Preconditions	The production activities have been scheduled and are involved in a time conflict.	
Success End Condition	The production activities have been shifted.	
Primary Actor **Secondary Actors**	Sales department. Resource units.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each production activity to be performed after the given production activity that is involved in a time conflict*: have the production activity shifted (**Use Case 14**).

Table 4-14. Extended use case for the business process "Have the production activity shifted".

USE CASE 14	Have the production activity shifted.	
Goal in Context	The production department expects the production activity to be shifted.	
Scope & Level	Production department, subfunction.	
Preconditions	The production activity has been scheduled and is involved in a time conflict.	
Success End Condition	The production activity has been shifted.	
Primary Actor **Secondary Actors**	Sales department. Resource units.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The production department sets the earliest start time of the production activity and sends to the corresponding resource unit a request to reschedule the production activity.
	2	The production department receives from the resource unit a confirmation of rescheduling of the production activity and registers the rescheduling.

Table 4-15. Extended use case for the business process "Delete the production order".

USE CASE 15	Delete the production order.	
Goal in Context	The sales department expects the production order and the corresponding product set to be deleted.	
Scope & Level	Sales department, primary task.	
Preconditions	The production order has been scheduled.	
Success End Condition	The production order and the corresponding product set have been deleted.	
Primary Actor **Secondary Actors**	Sales department. Resource units.	
Trigger	A request to delete the production order by the sales department.	
DESCRIPTION	**Step**	**Action**
	1	The production department has the production activities of the product set corresponding to the production order deleted (**Use Case 16**).
	2	The production department deletes the production order and the corresponding product set.

Table 4-16. Extended use case for the business process "Have the production activities deleted".

USE CASE 16	Have the production activities deleted.	
Goal in Context	The production department expects the production activities of the product set to be deleted.	
Scope & Level	Production department, subfunction.	
Preconditions	The production activities of the product set have been scheduled.	
Success End Condition	The production activities of the product set have been deleted.	
Primary Actor **Secondary Actors**	Sales department. Resource units.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each production activity of the product set*: delete the production activity and request the corresponding resource unit to delete it.

Table 4-17. Extended use case for the business process "Perform the production activity".

USE CASE 17	Perform the production activity.	
Goal in Context	The production department expects the production activity to be scheduled and performed.	
Scope & Level	Resource unit, primary task.	
Preconditions		
Success End Condition	The production activity has been scheduled and performed.	
Primary Actor **Secondary Actors**	Production department.	
Trigger	A request to schedule the production activity by the production department.	
DESCRIPTION	**Step**	**Action**
	1	The resource unit allocates the resources required for performing of the production activity to the production activity (**Use Case 18**).
	2	The resource unit registers the scheduling, commits towards the production department to complete the production activity, and informs the production department about the scheduling.
	3	*A signal by an internal worker on changing the capacity of a resource is received and the production activity is allocated to the resource having the capacity conflict*: resolve the capacity conflict (**Use Case 24**).
	4	*A signal on starting of the production activity by an internal worker is received*: register the start of the production activity (**Use Case 22**).
	5	*A signal on completion of the production activity by an internal worker is received*: register the end of the production activity (**Use Case 23**).

Table 4-18. Extended use case for the business process "Allocate the resources".

USE CASE 18	Allocate the resources.	
Goal in Context	The resource unit expects the resources required for performing of the production activity to be allocated to the production activity.	
Scope & Level	Resource unit, subfunction.	
Preconditions		
Success End Condition	The resources required for performing of the production activity have been allocated to the production activity.	
Primary Actor **Secondary Actors**	Production department.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each resource required for performing of the production activity*: allocate the resource to the production activity.

Table 4-19. Extended use case for the business process "Reschedule the production activity".

USE CASE 19	Reschedule the production activity.	
Goal in Context	The production department expects the production activity involved in a capacity conflict to be rescheduled.	
Scope & Level	Resource unit, primary task.	
Preconditions	The production activity has been scheduled and is involved in a capacity conflict.	
Success End Condition	The production activity has been rescheduled.	
Primary Actor Secondary Actors	Production department.	
Trigger	A request to reschedule the production activity has been received from the production department.	
DESCRIPTION	**Step**	**Action**
	1	The resource unit deletes the commitment of the resource unit towards the production department to complete the production activity.
	2	The resource unit deletes the resource allocations of the production activity (**Use Case 20**).
	3	The resource unit re-allocates the resources required for performing of the production activity to the production activity (**Use Case 18**).

Table 4-20. Extended use case for the business process "Delete the resource allocations".

USE CASE 20	Delete the resource allocations.	
Goal in Context	The resource unit expects the resource allocations of the production activity to be deleted.	
Scope & Level	Resource unit, subfunction.	
Preconditions	The production activity has been scheduled and is involved in a capacity conflict.	
Success End Condition	The resource allocations of the production activity have been deleted.	
Primary Actor Secondary Actors	Production department.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each resource allocated to the production activity*: delete the allocation.

Table 4-21. Extended use case for the business process "Delete the production activity".

USE CASE 21	Delete the production activity.	
Goal in Context	The production department expects the production activity to be deleted.	
Scope & Level	Resource unit, primary task.	
Preconditions	The production activity has been scheduled.	
Success End Condition	The production activity has been deleted.	
Primary Actor Secondary Actors	Production department.	
Trigger	A request by the production department to delete the production activity.	
DESCRIPTION	**Step**	**Action**
	1	The resource unit deletes the commitment of the resource unit towards the production department to complete the production activity.
	2	The resource unit deletes the resource allocations of the production activity (**Use Case 20**).
	3	The resource unit deletes the production activity.

Table 4-22. Extended use case for the business process "Register the start of the production activity".

USE CASE 22	Register the start of the production activity.	
Goal in Context	The resource unit expects the start of the production activity to be registered.	
Scope & Level	Resource unit, subfunction.	
Preconditions	The production activity has been scheduled.	
Success End Condition	The start of the production activity has been registered.	
Primary Actor Secondary Actors	Internal worker.	
Trigger	A signal on starting of the production activity by an internal worker.	
DESCRIPTION	**Step**	**Action**
	1	The resource unit registers the start of the production activity.

Table 4-23. Extended use case for the business process "Register the end of the production activity".

USE CASE 23	Register the end of the production activity.	
Goal in Context	The resource unit expects the end of the production activity to be registered.	
Scope & Level	Resource unit, subfunction.	
Preconditions	The production activity has been scheduled.	
Success End Condition	The end of the production activity has been registered.	
Primary Actor Secondary Actors	Internal worker.	
Trigger	A signal on completion of the production activity by an internal worker.	
DESCRIPTION	**Step**	**Action**
	1	The resource unit registers the end of the production activity, discharges the commitment of the resource unit towards the production department to complete the production activity, and informs the production department about the completion of the production activity.

Table 4-24. Extended use case for the business process "Resolve the capacity conflict".

USE CASE 24	Resolve the capacity conflict.	
Goal in Context	The resource unit expects all the production activities allocated to the resource having the capacity conflict to be rescheduled.	
Scope & Level	Resource unit, subfunction.	
Preconditions	A capacity conflict of the resource has been detected.	
Success End Condition	All the production activities allocated to the resource having the capacity conflict have been rescheduled.	
Primary Actor Secondary Actors	Internal worker.	
Trigger	A signal by an internal worker on changing the capacity of a resource.	
DESCRIPTION	**Step**	**Action**
	1	The resource unit cancels all the resource allocations of production activities to the resource having the capacity conflict (**Use Case 25**).
	2	The resource unit reschedules all the production activities that are to be allocated to the resource having the capacity conflict (**Use Case 26**).

Table 4-25. Extended use case for the business process "Cancel the resource allocations".

USE CASE 25	Cancel the resource allocations.	
Goal in Context	The resource unit expects all the resource allocations of production activities to the resource having the capacity conflict to be deleted.	
Scope & Level	Resource unit, subfunction.	
Preconditions	A capacity conflict of the resource has been detected.	
Success End Condition	All the resource allocations of production activities to the resource having the capacity conflict have been deleted.	
Primary Actor Secondary Actors	Internal worker.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each production activity allocated to the resource having the capacity conflict*: the resource unit deletes the resource allocations of the production activity (**Use Case 20**).

Table 4-26. Extended use case for the business process "Reschedule the production activities".

USE CASE 26	Reschedule the production activities.	
Goal in Context	The resource unit expects all the production activities to be allocated to the resource having the capacity conflict to be rescheduled.	
Scope & Level	Resource unit, subfunction.	
Preconditions	All the resource allocations of production activities to the resource having the capacity conflict have been deleted.	
Success End Condition	All the production activities to be allocated to the resource having the capacity conflict have been rescheduled.	
Primary Actor Secondary Actors	Internal worker.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each production activity to be allocated to the resource having the capacity conflict*: schedule the production activity (**Use Case 17**).

4.1.5. Design By Extended AOR Modelling

4.1.5.1. Organization Modelling

According to [Ow88], production scheduling decisions for large and/or complex manufacturing facilities are often not made by any single individual. Rather, a group of people may be identified in the organization who cooperate and share information to develop and manage a production schedule. Because of the routine nature of the scheduling task, this group has usually adopted some organization structure to make the decision-making process efficient. As a drastic change to this organization structure is not desired, the AORML agent diagram of the ceramic factory's organization structure, shown in Figure 4-1, reflects the existing factory. However, the organization structure has been complemented with some generalizations in line with the principles of reactive scheduling introduced in section 4.1.3.

The agent instance CeramicFactory depicted in Figure 4-1, which belongs to the agent type Organization, represents the ceramic factory to be modelled. There is an isBenevolentTo relationship between the internal agent type SalesDepartment of the CeramicFactory and the external agent type Customer. The agent CeramicFactory consists of instances of the following subclasses of the institutional agent type OrganizationUnit, representing departments and other internal units of the factory: FactoryManagement, TechnologicalDepartment, AccountingDepartment, SalesDepartment, ProductionDepartment, DesignDepartment, SupplyDepartment, and ResourceUnit. Like in section 3.8.1, the number of instances of an agent type is shown in the top right corner of the box with rounded corners denoting the corresponding agent type.

The institutional agent type FactoryManagement includes the internal institutional agent type ManagingBoard. In the same way, the institutional agent type SalesDepartment includes the internal institutional agent types FactoryShop and CompletedProductionStore, and the institutional agent type SupplyDepartment includes the institutional agent type RawMaterialStore. In addition to other institutional agent types, the institutional agent types modelled in Figure 4-1 include human role types, like, e.g., ChiefTechnologist, Accountant, and Designer. A typical pattern of internal agent types within an institutional agent type representing a department of the factory consists of the human role type of the head of the department, like ChiefTechnologist, ChiefAccountant, SalesManager, ChiefDesigner, and SupplyManager, who has one or more human role types subordinated to it.

Within the ceramic factory, like within any other organization, there is a hierarchy of roles where one role is subordinate to another role. For example, in Figure 4-1 there is an isSubordinateTo relationship between the human role types BoardMember and StaffManager on one hand and the human role type ManagingDirector on the other. The human role type BoardMember forms a superclass of the human role types SalesManager and ProductionManager. The human role types ManagingDirector and BoardMember are included by the internal institutional agent type ManagingBoard which reflects the fact that the ManagingDirector as well as the SalesManager and ProductionManager belong to the managing board of the factory by virtue of their offices. There is also an isSubordinateTo relationship between several other human role types in Figure 4-1, like, e.g., between the internal agent types StoreKeeper and SalesManager of SalesDepartment. All the human role types represented in Figure 4-1 are subclasses of the human role type EmployeeOfCeramicFactory, which, in turn, forms a subclass of the agent type Person. For the sake of clarity of Figure 4-1, these agent types are not represented in the figure.

The Order Scheduler and Resource Scheduler introduced in section 4.1.3 are represented in Figure 4-1 by the institutional agent types ProductionDepartment and ResourceUnit which respectively consist of the human role types ProductionManager and Worker. Instances of Worker are subordinated to the instance of ProductionManager. The institutional agent type ResourceUnit has been introduced to enable the modelling of two-perspective scheduling described in section 4.1.3. In the ceramic factory to be modelled there is no real agent type corresponding to it, even though workers effectively form teams according to their specialties. The institutional agent type ResourceUnit has as subclasses the institutional agent types MouldmakingUnit, Moulding/CastingUnit, ElaborationUnit, BurningUnit, RawMaterialStore, and CompletedProductionStore, representing different types of resource units of the ceramic factory.

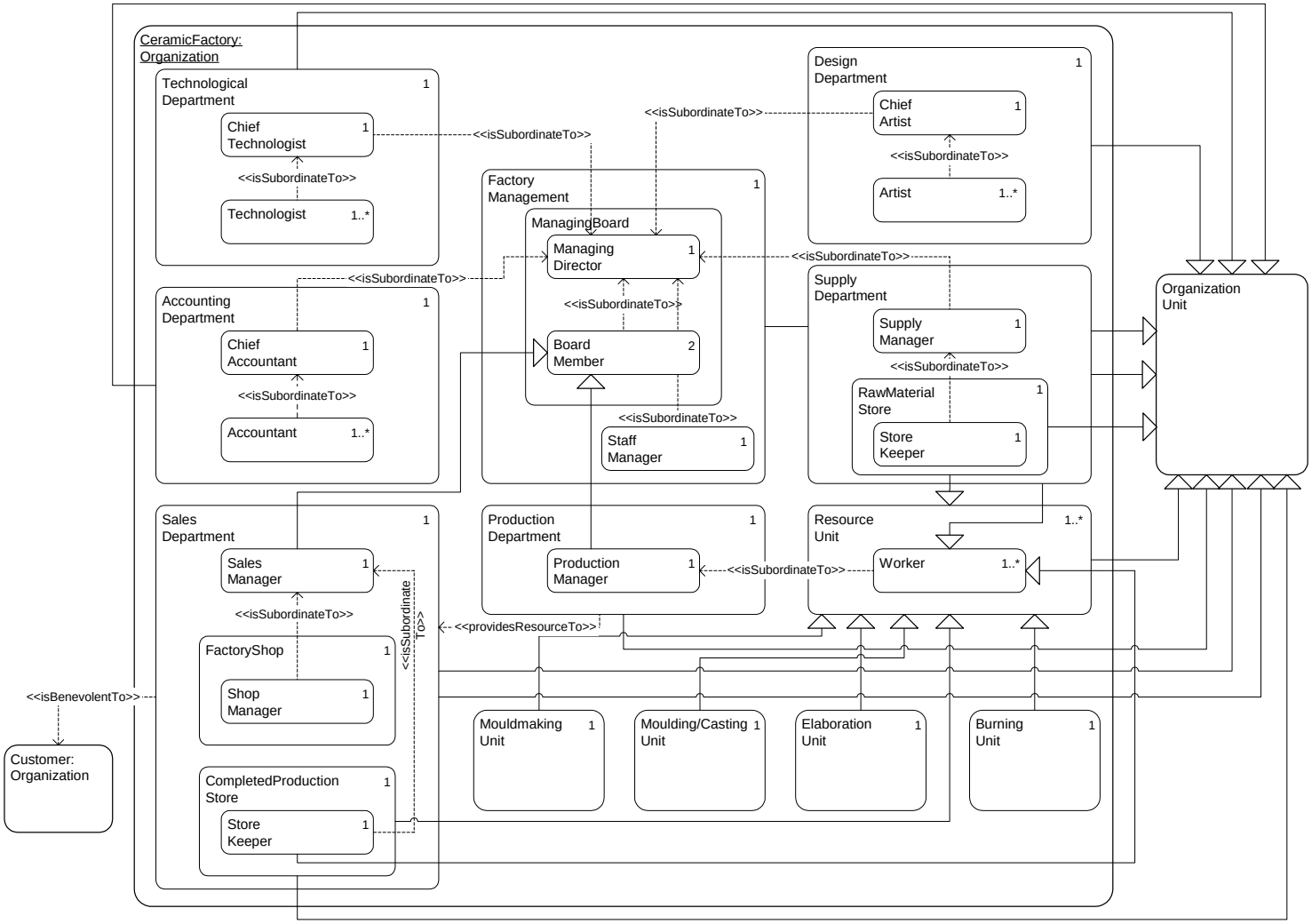


Figure 4-1. The organization model of the ceramic factory.

4.1.5.2. Information Modelling

Principles of Creating Scheduling Ontologies

The OZONE scheduling ontology described in [Smith97] can be described as a meta-model of the domain of scheduling. It provides a language for describing those aspects of the scheduling domain that are relevant to construction of an application system, and a set of constraints on how concepts in the language fit together to form consistent domain models. Consistency, in this context, relates to the information and knowledge required to insure executability of the model. Generally speaking, the ontology serves to map user-interpretable descriptions of an application domain to application system functionality [Smith97]. As such, the principles underlying the OZONE ontology can be applied to the modelling and simulation of the ceramic factory's business processes from the point of view of scheduling and managing of schedules.

Scheduling is defined in OZONE as a process of feasibly synchronizing the use of RESOURCES by ACTIVITIES to satisfy DEMANDS over time, and application problems are described in terms of this abstract domain model. Figure 4-2, adopted from [Smith97], illustrates the base concepts involved and their structural relationships. A DEMAND is an input request for one or more PRODUCTS, which designate the GOODS or SERVICES required. Satisfaction of DEMANDS centers around the execution of ACTIVITIES. An ACTIVITY is a process that uses RESOURCES to produce goods or provide services. The use of RESOURCES and the execution of ACTIVITIES are restricted by a set of CONSTRAINTS. These five base concepts of the ontology – DEMAND, ACTIVITY, RESOURCE, PRODUCT, and CONSTRAINT – together with the inter-relationships depicted in Figure 4-2, define an abstract model of a scheduling domain, and a framework for analyzing and describing particular application environments. Associated with each concept definition are terminologies for describing basic properties and capabilities. Properties define attributes or parameters of relevance to specifying an executable scheduling model [Smith97]. Capabilities roughly correspond to *methods* of object types (i.e., of concepts) related to the scheduling functionality, such as *Find-Schedulable-Time* associated with an ACTIVITY.

According to [Smith97], plans and schedules are represented as networks of ACTIVITIES in OZONE, with an ACTIVITY containing various decision variables (e.g., start time, end time, assigned resources). To construct a schedule that satisfies a given input DEMAND, it is necessary to first instantiate a set of ACTIVITIES that will produce (provide) the designated PRODUCT.

To schedule an ACTIVITY, it is necessary to choose specific RESOURCES, which involves determining intervals where resources have capacity available to support execution of the ACTIVITY, and subsequently allocating capacity of chosen RESOURCES to ensure that they will not be used by other ACTIVITIES. The semantics of allocating (and de-allocating) resource capacity varies according to the type of RESOURCE involved. To this end, a RESOURCE maintains a representation of its available capacity over time [Smith97].

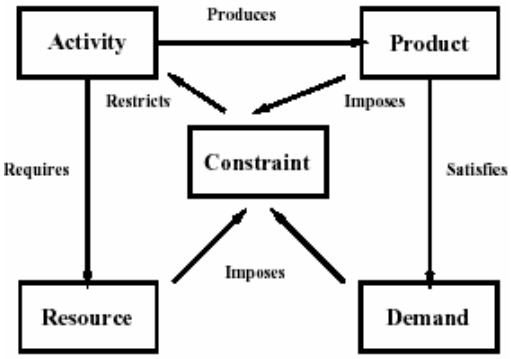


Figure 4-2. Abstract domain model of the OZONE ontology.

The Scheduling Ontology of the Ceramic Factory

The domain model of the ceramic factory depicted in Figure 4-3 is based on the principles of creating scheduling ontologies that were presented in the previous subsection. In Figure 4-3, the institutional agent CeramicFactory includes (instances of) the institutional internal agent types FactoryManagement, SalesDepartment, ProductionDepartment, and ResourceUnit. For the latter the subclasses MouldmakingUnit, RawMaterialStore, and CompletedProductionStore are shown in Figure 4-3. All of them were introduced in section 4.1.5.1. The agents have knowledge/information about their proprietary object types and about object types that are shared between agents of different types.

In the ontology depicted in Figure 4-3, the concept DEMAND of Figure 4-2 is represented by the object type ProductionOrder. It is shared between the agent CeramicFactory and the agent type Customer. A ProductionOrder is characterized by a number of attributes, the most important ones of which are releaseTime, dueTime, productCode, and quantity, and by the shared status predicate isCompleted. The attributes releaseTime and dueTime are respectively the earliest and latest time when the production activities for producing the product set corresponding to the ProductionOrder can start and end, respectively. The attributes productCode and quantity respectively specify the type and number of the products in the product set requested. The internal representation of the object type ProductionOrder within the agent CeramicFactory satisfies one of the following status predicates: isPreliminary, isScheduled, isProposed, isAccepted, isRejected, or isDelivered.

There is a shared object type Invoice connected to the informational entity types ProductionOrder, Customer, and CeramicFactory. It includes a number of attributes like orderID, productCode, quantity, price, subtotal, VAT, and total. In addition, its internal representation within the SalesDepartment is possessed of the status predicates isPreliminary, isSent, and isPaid.

In Figure 4-3, the type of the products requested is modelled by the association between the object types ProductionOrder and ProductType. An instance of ProductType is identified by its attributes productName (e.g., "coffee cup Kirke") and productCode (e.g., "22882"). The internal representation of the object type ProductType within the agent type ProductionDepartment differs from its base object type by a number of relations to other object types. Among them, an ordered sequence of instances of ProductionActivityType associated with an instance of ProductType define the product type in question. Specific products to be produced to satisfy production orders are represented as instances of ProductSet which corresponds to the concept PRODUCT in Figure 4-2. Each ProductSet references an ordered sequence of instances of ProductionActivity which form the set of processing steps required to produce the ProductSet. There are associations of the type PrecedenceInterval between instances of ProductionActivityType. Each association specifies the lower bound and upper bound of the temporal separation between production activities of two types. In conformance with [Smith89], the associations of the type PrecedenceInterval are intended to provide a basis for describing generic manufacturing processes, defining sets of possible ProductionActivity sequences. Precedence relations between instantiated production activities, i.e. sequences of production activities that are known with certainty, are derived from possible sequences and instances of ProductionActivity by means of derivation rules provided in Appendix D.

As well as the OZONE ontology briefly described in the previous section, the scheduling ontology of the ceramic factory adopts an activity-centered modelling viewpoint. In the center of the ontology represented in Figure 4-3 is thus the shared object type ProductionActivity, corresponding to the concept ACTIVITY in Figure 4-2. An object of the type ProductionActivity can have the status isUnscheduled, isScheduled, isInProcess, or isCompleted. An instance of ProductionActivity is characterized by the following attributes: activityID, typeName, earliestStartTime, quantity, startTime, and endTime. The identifier attribute activityID contains the identifier of the production activity which is automatically assigned to it upon creation of the corresponding object. The attribute typeName repeats the value of the attribute activityName of the activity's ProductionActivityType which is discussed below. The action of scheduling a ProductionActivity results in determining values for the attributes startTime and endTime. The attribute earliestStartTime indicates the earliest time at which the given ProductionActivity can be performed, considering the endTime of the previous activity scheduled or the releaseTime of the ProductionOrder in case of the first production activity. Each instance of ProductionActivity belongs to some ProductionActivityType which is represented by the corresponding many-to-one relationship in Figure 4-3. An instance of ProductionActivityType is characterized by the name of the activity type (activityName) and the average speed of performing an activity of the corresponding type (numberOfProductsPerHour). The latter includes the time required for setting up the resources before a

ProductionActivity of the given type can actually start. The object type ProductionActivity has a specific internal representation within the agent type ProductionDepartment. It complements the status predicate isScheduled with the internal intensional predicate hasTimeConflict(ProductionOrder) because a time conflict between scheduled activities is always detected within the ProductionDepartment. The intensional predicate hasTimeConflict (ProductionOrder) can be expressed as the following OCL operation attached to the object type ProductionActivity where the helper operation getEarliestStartTime (ProductionOrder) of ProductionActivity is defined in Appendix D:

context ProductionActivity::hasTimeConflict (order : ProductionOrder) : Boolean
post: **result** = (**self**.getEarliestStartTime(order) > **self**.startTime)

Also important in a scheduling ontology is the concept of a resource. It is represented as a shared object type Resource in Figure 4-3 which corresponds to the concept RESOURCE in Figure 4-2. Each institutional agent of the type ResourceUnit has knowledge about objects of at least one of the proprietary subtypes ReusableResource and DiscreteStateResource of Resource. A ReusableResource, like a set of ceramic moulds, is a resource whose capacity becomes available for reuse after the ProductionActivity to which it has been allocated finishes. An instance of ReusableResource is characterized by two attributes: its cumulativeUsageTimes, which is the total amount of resource uses permitted (e.g., the number of times that the use of a MouldSet is permitted for moulding or casting), and the numberOfResources. A DiscreteStateResource, like a worker or a machine or a combination of them, is a resource whose availability is a function of some discrete set of possible state values (e.g., *idle* and *busy*). Each instance of DiscreteStateResource consists of the internal object :Capacity and instances of the internal object type CapacityInterval. The Capacity specifies the numberOfResources and batchSize. The latter is the number of products that the resource can process simultaneously. The capacity of a resource is represented as an ordered sequence of instances of CapacityInterval, e.g., work shifts, with each interval indicating the instances of ProductionActivity that are anticipated to be consuming capacity within its temporal scope and the capacity that remains available [Smith89]. For each CapacityInterval, the startTime and endTime of the interval are thus specified. The specializations of CapacityInterval, not shown in Figure 4-3, are WorkMonth, WorkWeek, and WorkShift. They are present in the simulation environment of the ceramic factory that will be briefly described in section 4.1.5.6. Successful scheduling results in attaching a CapacityInterval to one or more instances of ProductionActivity. For determining whether a CapacityInterval can be allocated to the given ProductionActivity, the object type CapacityInterval has the intensional predicate isSchedulable (ProductionActivity). There are two versions of this intensional predicate because the allocation of capacity intervals of a DiscreteStateResource to production activities is different for instances of its two subclasses UnitCapacityResource and BatchCapacityResource. An instance of UnitCapacityResource, like a worker, can process only one product at a time, i.e., its batchSize is 1, whereas a BatchCapacityResource, like a kiln, can process simultaneously up to batchSize products. This is also reflected by the respective two subclasses of CapacityInterval: UnitCapacityInterval and BatchCapacityInterval. While the available capacity of a UnitCapacityInterval is characterized by the attribute availableDuration (e.g., per work shift), the available capacity of a BatchCapacityInterval is represented by the number of products (availableCapacity) that the resource (e.g., a kiln) is capable of processing at a time. The intensional predicates isSchedulable(ProductionActivity) for UnitCapacityInterval and BatchCapacityInterval are defined in Appendix D. The first of them looks like as follows:

context UnitCapacityInterval::isSchedulable (a : ProductionActivity) : Boolean =
a.earliestStartTime <= endTime **and** requiredDuration(a) <= availableDuration **and**
self.unitCapacityResource.unitCapacityInterval->select(a.earliestStartTime <= endTime **and**
requiredDuration(a) <= availableDuration)->forAll(**self**.startTime <= startTime)

The intensional predicate above makes use of the helper operation requiredDuration(ProductionActivity) of UnitCapacityInterval which calculates the time (in minutes) that the given UnitCapacityResource needs for performing the ProductionActivity. The select-operation of OCL returns all the capacity intervals that can be allocated to the instance of ProductionActivity referred to by a, and the forAll-operation following it makes sure that only the earliest capacity interval among them is allocated to the ProductionActivity.

All the resource types represented in Figure 4-3 are human resources. According to [Smith90], ***human resources*** are resources that comprise the human work done. Depending on the characteristics of a given manufacturing environment, human resources might be defined as either *primary* or *secondary resources* from the standpoint of allocation. For example, a burner is a secondary resource

because its allocation accompanies the allocation of a resource of another type – kiln, while a caster is a primary resource.

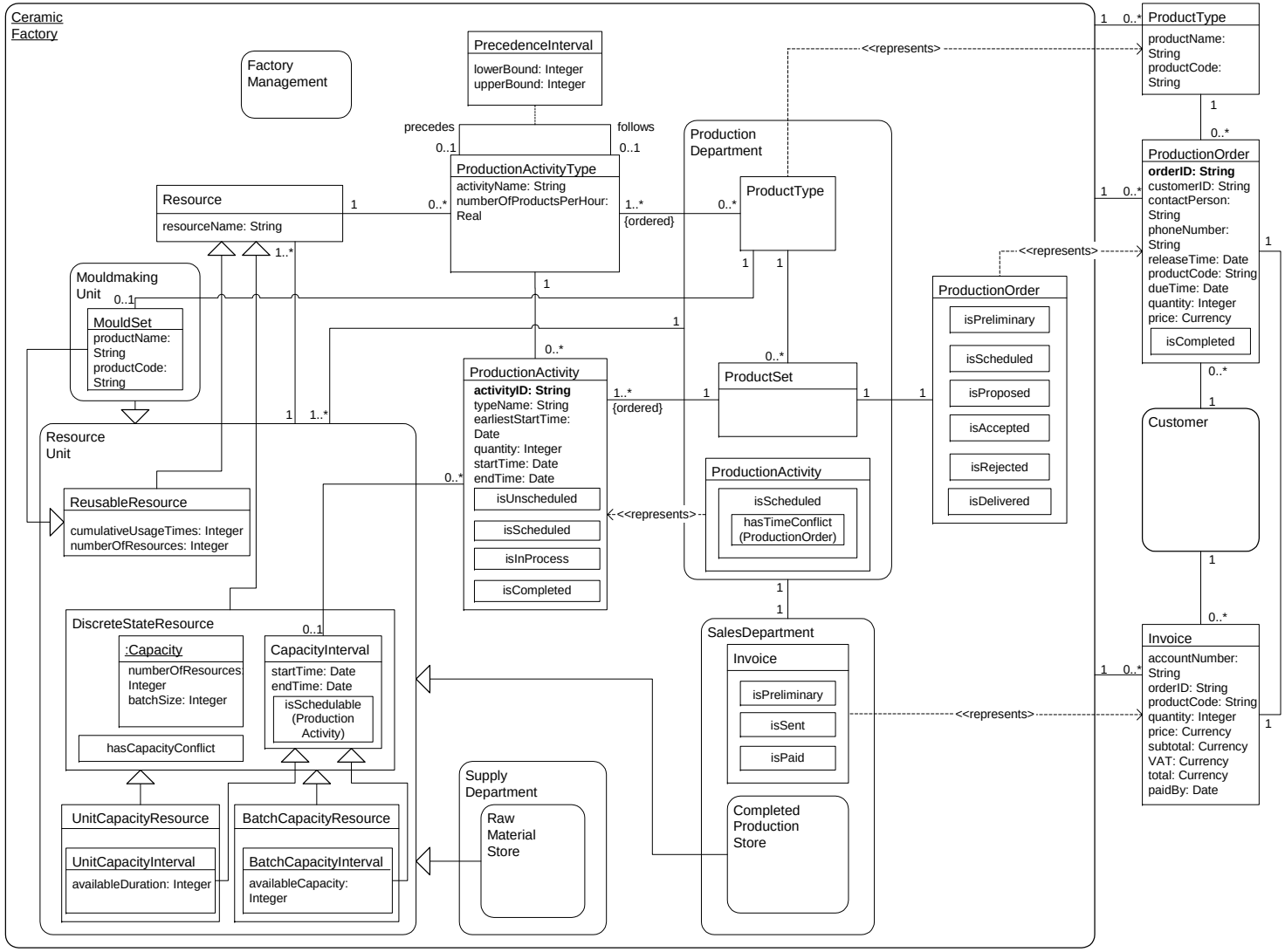


Figure 4-3. The domain model of the ceramic factory.

4.1.5.3. Interaction Modelling

The interaction frames in the extended interaction frame diagram depicted in Figure 4-4 correspond to the use cases represented in Tables 4-1 – 4-26.

The first communicative action event type in the interaction frame between the agent type Customer and the internal agent type SalesDepartment of the CeramicFactory represents a request by the Customer to provide it with the product set which is identified by the product code (?String) and quantity needed (?Integer). Since there is an isBenevolentTo relationship between the agent types SalesDepartment and Customer in the organization model of Figure 4-1, the next three communicative action event types model a proposal by the SalesDepartment to provide the Customer with the product set according to the production order created by the SalesDepartment, and its acceptance or rejection by the Customer. The instance of the production order, which includes a specific due time, is described by the data element ?ProductionOrder of the corresponding communicative action event. If the proposal is accepted, the SalesDepartment commits on behalf of the CeramicFactory towards the Customer to provide it with the product set corresponding to the production order. A commitment/claim of this type is satisfied by an action event of the type provideProductSet(?ProductionOrder) which is coupled with the corresponding commitment/claim type. The next action event types are used after the product set has been produced. The SalesDepartment first informs the Customer about the completion, and the latter then issues to the CompletedProductionStore (an internal institutional agent of the SalesDepartment) a request to release the product set identified by the corresponding ProductionOrder. The CompletedProductionStore provides the Customer with the product set in question and also sends to the Customer the invoice (?Invoice). This creates for the SalesDepartment a claim against the Customer that it would pay for the product set according to the invoice. The claim is satified by actual paying for the product set by the Customer.

The starting point for creating the interaction frame between the internal agent types SalesDepartment and ProductionDepartment is the providesResourceTo relationship between them in Figure 4-1. The ProductionDepartment thus provides the SalesDepartment with the resources needed for selling the products of the factory. The first communicative action event type of the interaction frame models a request by the SalesDepartment to the ProductionDepartment to schedule the production order described by the action event's data element ?ProductionOrder. Since neither scheduling a production order nor producing a product set according to it can be immediately perceived by the SalesDepartment, both are represented in terms of the domain model of the ceramic factory shown in Figure 4-3 as making true the respective status predicates isScheduled and isCompleted of the corresponding instance of ProductionOrder. After the ProductionDepartment has returned the scheduled production order to the SalesDepartment, it receives from the SalesDepartment a request to either have the production order completed or to delete it which is reflected by the corresponding communicative action event types. In the first case, a *stit*-commitment/claim of the type achieve(isCompleted(?ProductionOrder)) is formed between the ProductionDepartment and SalesDepartment. The satisfaction of this commitment/claim is expressed by an instance of the corresponding achieve construct type.

The interaction frame between the agent types ProductionDepartment and ResourceUnit in Figure 4-4 is largely determined by the isSubordinateTo relationship between their internal agent types ProductionManager and Worker which is reflected at the level of their comprising organization units. This means that a ResourceUnit schedules and performs a production activity as requested by the ProductionDepartment and reports to the latter. The first communicative action event type between the agent types ProductionDepartment and ResourceUnit models a request by the ProductionDepartment to schedule the production activity that is described by the action event's data element ?ProductionActivity. In addition to initial scheduling of a production activity, a message of this type is also sent if a time conflict in the schedule is detected within the ProductionDepartment. The second message type models the confirmation of a scheduling by the ResourceUnit. The third message type, representing a request to delete the scheduled production activity described by ?ProductionActivity, is used only if the scheduled production order, which includes the production activity to be deleted, has been rejected by the Customer. Since the completion of a production activity cannot be directly perceived, it is modelled through the achieve construct type achieve(isCompleted(?ProductionActivity)) between the agent types ResourceUnit and ProductionDepartment. The achieve construct type is coupled with the corresponding *stit*-commitment/claim type because the completion of a production activity is preceded by the formation of the corresponding commitment/claim of this type. Communicative action event types inform(isScheduled(?ProductionActivity)), inform(isInProcess(?ProductionActivity)), and inform(isCompleted (?ProductionActivity)) are for informing the ProductionDepartment about the status changes of the production activity described by ?ProductionActivity.

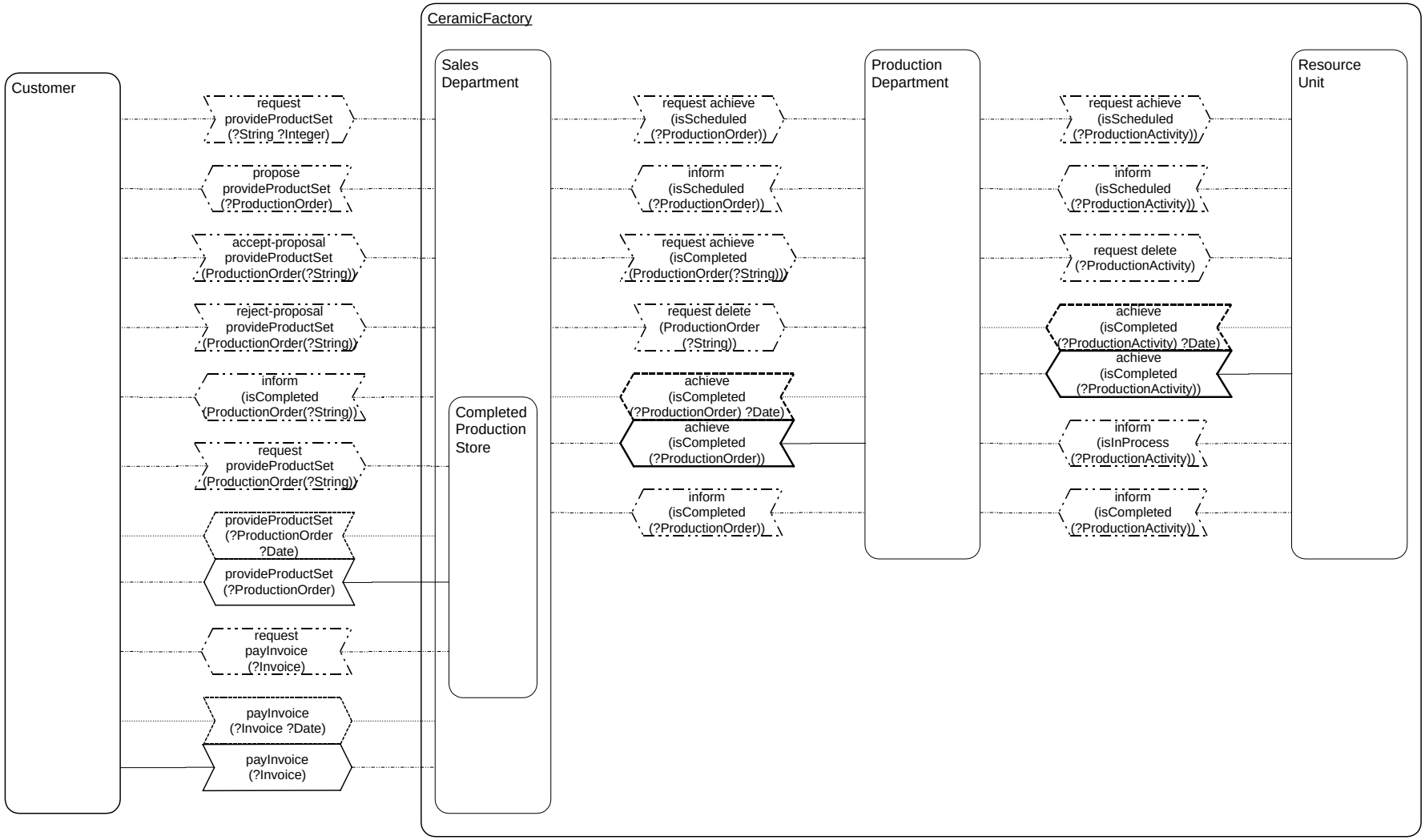


Figure 4-4. The extended interaction frame diagram of the CeramicFactory.

4.1.5.4. Function and Goal Modelling

By following guideline 1 of the recursive procedure described in section 3.8.4.1, main scenarios of the descriptions of the business/manufacturing processes of the ceramic factory by goal-based use cases presented in Tables 4-1 – 4-26 are turned into the corresponding activity types of the proper agent types. In Appendix F, the activity types distinguished at the stage of function and motivation modelling are presented in bold.

Firstly, the main scenarios of the primary tasks triggered by external agents are modelled according to guideline 1 in section 3.8.4.1. For example, the main scenarios of use cases 1 ("Have the product set produced") and 5 ("Have the product set delivered") with the sales department in focus are represented in Appendix F as the respective activity types "Manage production order" and "Manage product delivery" of the internal agent type SalesDepartment of the CeramicFactory. Analogously, the main scenarios of use cases 7 ("Create the product set and schedule the production order") and 11 ("Complete the production order") with the production department in focus are turned in Appendix F into the respective activity types "Process production order" and "Complete production order" of the internal agent type ProductionDepartment. According to guideline 2 in section 3.8.4.1, the triggers of the primary tasks mentioned are modelled in Appendix F as the respective reaction rules R1, R11, R19, and R27. The main scenarios of use cases 17 ("Perform the production activity") and 19 ("Reschedule the production activity") with the resource unit in focus are represented in Appendix F as the respective activity types "Schedule and perform production activity" and "Reschedule production activity" of the internal agent type ResourceUnit. The triggers of these primary tasks are modelled in Appendix F as the respective reaction rules R44 and R48.

Next, subordinate use cases (subfunctions) of the primary tasks are turned into the respective subactivity types of the activity types modelled as is described in guidelines 3 – 5 in section 3.8.4.1. The subfunctions "Have the production order scheduled" (Use Case 2), "Make a proposal and process the reply" (Use Case 3), and "Have the production order completed" (Use Case 4) of the primary task "Have the product set produced" (Use Case 1) are thus modelled as the respective sequential subactivity types "Have production order scheduled", "Manage proposal", and "Manage completion" of the internal agent type SalesDepartment of the CeramicFactory.

Following the same guidelines, the activity type "Process production order" of the internal agent type ProductionDepartment, corresponding to the primary task "Create the product set and schedule the production order" (Use Case 7), is refined into the subactivity types "Instantiate production plan" and "Schedule production order". The subfunction "Follow the production activities" (Use Case 12) of the primary task "Complete the production order" (Use Case 11) is represented as the corresponding subactivity type "Follow production activities" of the activity type "Complete production order" pertaining to the internal agent type ProductionDepartment. The activity type "Follow production activities" is, in turn, refined into the subactivity type "Have production activities shifted".

Analogously, the subfunctions "Allocate the resources" (Use Case 18), "Register the start of the production activity" (Use Case 22), "Register the end of the production activity" (Use Case 23), and "Resolve the capacity conflict" (Use Case 24) of the primary task "Perform the production activity" (Use Case 17) are represented in Appendix F as the respective subactivity types "Allocate resources", "Register the start of the production activity", "Register the end of the production activity", and "Resolve the capacity conflict" of the internal agent type ResourceUnit. The last three subactivity types mentioned are triggered by an internal agent of the type Worker of ResourceUnit. The triggers of these activity types are modelled as the respective reaction rules R58, R60, and R62. Still according to guidelines 3 - 5 in section 3.8.4.1, the activity type "Reschedule production activity" of the internal agent type ResourceUnit, corresponding to the use case "Reschedule the production activity" (Use Case 19), is modelled as consisting of the subactivity types "Delete allocations" and "Allocate resources". The subactivity type "Delete allocations" is also included by the activity type "Delete production activity and commitment" which corresponds to the primary task "Delete the production activity" (Use Case 21).

The process described by guidelines 3 - 5 in section 3.8.4.1 is recursively repeated for all subfunctions and steps of a primary task as long as the modelling precision of the desired level is achieved. If the step of a scenario does not include any subfunctions, it is modelled as a sequential elementary activity type. For example, step 1 of the primary task "Have the product set produced" (Use Case 1) and steps 1 and 4 of the primary task "Create the product set and schedule the production order" (Use Case 7) are modelled as the elementary activity types "Create production order" of

SalesDepartment, and "Create product set" and "Send scheduled production order" of ProductionDepartment. The subfunctions "Register the start of the production activity" (Use Case 22) and "Register the end of the production activity" (Use Case 23) are modelled as the elementary activity types of the same names of the internal agent type ResourceUnit.

In function and motivation models we do not represent types of activities that are repeated for each instance of some (possibly constrained by a predicate) informational entity type because activity diagrams of function and motivation models do not lend themselves to expressing conditions (of repeating). For example, as Appendix F shows, the activity types "Instantiate production activity" and "Have production activity scheduled" of the internal agent type ProductionDepartment, whose instances are repeated for each instance of ProductionActivity, are not represented in function and motivation models.

As we stated in section 3.8.4.2, preconditions and goals are defined for activity types as propositions by means of OCL. In Table 4-27, the precondition and goal defined for each activity type of the function model of the case study of the ceramic factory is presented.

The goal defined for the activity type "Manage production order" of the internal agent type SalesDepartment means that the product set specified by the customer using the product code and quantity required has been produced and the SalesDepartment has a commitment towards the Customer to provide it with the product set by the due time specified in the instance of ProductionOrder which has been returned to the Customer. The commitment is satisfied within an activity of the type "Manage product delivery" whose precondition states that the instance of ProductionOrder referred to by the value of the input parameter order exists and has the status isCompleted. The status predicate isCompleted is defined in Appendix D. The meaning of the goal defined for the activity type "Manage product delivery" is that the product set corresponding to the production order, which is referred to by the value of the activity's input parameter order of the type ProductionOrder, has been delivered, the commitment has been satisfied (i.e., does not exist any more), and there exists the corresponding invoice that has been sent to the customer.

Analogously, the precondition defined for the activity type "Process production order" of the internal agent type ProductionDepartment is the existence of the instance of ProductionOrder with the status isPreliminary that is referred to by the value of the activity's input parameter order and is associated with the instance of ProductType identified by the value of the production order's attribute productCode. The association mentioned is created by reaction rule R19 in Appendix F upon receiving the production order from the SalesDepartment. The goal defined for the activity type "Process production order" means that there exists the instance of ProductSet, corresponding to the instance of ProductionOrder, and that the production order has been scheduled. According to the derivation rule defined in Appendix D, an instance of ProductionOrder has the status isScheduled if every instance of ProductionActivity of the corresponding ProductSet has the status isScheduled.

The precondition defined for the activity type "Complete production order" is that within the corresponding agent of the type ProductionDepartment exists the instance of ProductionOrder with the status isScheduled that is referred to by the value of the activity's input parameter order. The goal defined for the same activity type expresses that the product set specified by the production order has been produced which is reflected by the status isCompleted of the ProductionOrder. The status predicate isCompleted of the object type ProductionOrder is defined in Appendix D.

The precondition defined for the activity type "Schedule production activity" of the internal agent type ResourceUnit is the existence of the instance of ProductionActivity with the status isPreliminary that is referred to by the value of the activity's input parameter activity and is associated with the instance of ProductionActivityType identified by the production activity's attribute activityTypeName. The association mentioned is created by reaction rule R44 in Appendix F upon receiving the production activity from the ProductionDepartment. The meaning of the goal defined for the activity type "Schedule production activity" is that the corresponding production activity is scheduled and the ResourceUnit has the *see-to-it-that* commitment towards the ProductionDepartment to complete the production activity by its scheduled end time. The status predicate isScheduled of the object type ProductionActivity is defined in Appendix D.

For the production activity type "Reschedule production activity", the precondition requires the value of the input parameter activity to refer to an instance of ProductionActivityType with the status isScheduled. The goal defined for the activity type mentioned is the same as that defined for the activity type "Schedule production activity".

Table 4-27. Activities of the case study of the ceramic factory with their preconditions and goals.

Activity type and input parameter(s)	**Precondition**	**Goal**
Manage production order (code : String, quant : Integer, senderID : String)	-	ProductionOrder.allInstances->exists (o : ProductionOrder | o.isCompleted and o.productCode = code and o.quantity = quantity and provideProductSet.allInstances->exists (about = o and dueTime = o.dueDate and sourceID = self.agentID and targetID = senderID))
Create production order	-	ProductionOrder.allInstances->exists (o : ProductionOrder | o.isPreliminary and o.productCode = code and o.quantity = quantity)
Manage scheduling and completion (order : ProductionOrder)	ProductionOrder.allInstances->exists (o : ProductionOrder | o.isPreliminary and o.productCode = code and o.quantity = quantity and order = o)	order.isCompleted **and** provideProductSet.allInstances->exists (about = order and dueTime = order.dueDate and sourceID = self.agentID and targetID = senderID)
Have production order scheduled	-	order.isScheduled
Request scheduling	-	order.isPreliminary
Register scheduling	-	order.isScheduled
Manage proposal	-	order.isAccepted **and** provideProductSet.allInstances->exists (about = order and dueTime = order.dueDate and sourceID = self.agentID and targetID = senderID)
Authorize and send proposal	-	order.isProposed
Manage completion	-	order.isCompleted
Request completion	-	-
Register completion	-	order.isCompleted
Inform the customer	-	-
Manage product delivery (order : ProductionOrder, senderID : String)	ProductionOrder.allInstances->exists (o : ProductionOrder | o.isCompleted and order = o)	order.isDelivered **and** **not**(provideProductSet.allInstances->exists (about = order and dueTime = order.dueDate and sourceID = self.agentID and targetID = senderID)) **and** Invoice.allInstances->exists(i : Invoice | i.isSent and i.orderID = order.orderID and order.invoice = i and i.productionOrder = order)
Deliver product set	-	order.isDelivered and **not**(provideProductSet.allInstances->exists (about = order and dueTime = order.dueDate and sourceID = self.agentID and targetID = senderID))
Create invoice	-	Invoice.allInstances->exists (i : Invoice | i.isPreliminary and i.orderID = order.orderID and order.invoice = i and i.productionOrder = order)
Send invoice	-	order.invoice.isSent

Table 4-27 (continued). Activities of the case study of the ceramic factory with their preconditions and goals.

Create claim	-	payInvoice.allInstances->exists (about = order.invoice and dueTime = order.invoice.paidBy and sourceID = senderID and targetID = self.agentID)
Register payment (invoice : Invoice, originID : String)	Invoice.allInstances->exists (i : Invoice | i.isSent and invoice = i)	invoice.isPaid and **not**(payInvoice.allInstances->exists (about = invoice and dueTime = invoice.paidBy and sourceID = originID and target ID = self.agentID))
Process production order (order : ProductionOrder)	ProductionOrder.allInstances->exists (o : ProductionOrder | o.isPreliminary and o.productType = ProductType.allInstances->any (pt: ProductType | pt.productCode = o.productCode and pt.productionOrder->includes(o)) and order = o)	ProductSet.allInstances->exists (ps : ProductSet | ps.productionOrder = order and order.productSet = ps and ps.productType = order.productType and order.productType->includes(ps)) **and** order.isScheduled
Create product set	-	ProductSet.allInstances->exists (ps : ProductSet | ps.productionOrder = order and order.productSet = ps and ps.productType = order.productType and order.productType->includes(ps))
Instantiate production plan	-	order.isPreliminary **and** order.productType.productionActivityType->forAll(t : productionActivityType | t.productionActivity->exists (a: ProductionActivity | a.isUnscheduled and a.typeName = t.activityName and a.productionActivityType = t and order.productSet->includes(a) and a.productSet = order.productSet))
Schedule production order	-	order.isScheduled
Send scheduled production order	-	-
Complete production order (order : ProductionOrder)	ProductionOrder.allInstances->exists (o : ProductionOrder | o.isScheduled and order = o)	order.isCompleted
Follow production activities (order : ProductionOrder)	-	order.isCompleted
Have production activities shifted (a : ProductionActivity)	ProductionActivity.allInstances->exists (pa : ProductionActivity | (pa.isScheduled or pa.isCompleted) and a = pa)	**not**(ProductionActivity. allInstances->exists (productSet.order = order and hasTimeConflict(order)))
Delete production order and product set (order : ProductionOrder)	ProductionOrder.allInstances->exists (o : ProductionOrder | o.isScheduled and order = o)	**not**(ProductSet.allInstances->exists(productionOrder = order and productType = order.productType)) **and** **not**(ProductionOrder.allInstances->exists (o : ProductionOrder | o = order))
Have production activities deleted	-	**not**(ProductionActivity. allInstances->exists (productSet.order = order))
Delete product set	-	**not**(ProductSet.allInstances->exists(productionOrder = order and productType = order.productType))

Table 4-27 (continued). Activities of the case study of the ceramic factory with their preconditions and goals.

Delete production order	-	**not**(ProductionOrder.allInstances->exists (o : ProductionOrder | o = order))
Schedule production activity (activity : ProductionActivity)	ProductionActivity.allInstances->exists (a : ProductionActivity | a.isUnscheduled and a.productionActivityType = ProductionActivityType->any (t: ProductionActivityType | t.activityName = a.typeName and t.productionActivity->includes(a)) and activity = a)	activity.isScheduled **and** self.stitCommitmentClaim->exists (achieve = activity.isCompleted and dueTime = activity.endTime and sourceID = self.agentID and targetID = senderID)
Allocate resources	-	activity.productionActivityType. discreteStateResource->forAll (capacityInterval->exists (ci : CapacityInterval | ci.productionActivity->includes(activity) and activity.capacityInterval = ci))
Reschedule production activity (activity : ProductionActivity)	ProductionActivity.allInstances->exists (a : ProductionActivity | a.isScheduled and activity = a)	activity.isScheduled **and** self.stitCommitmentClaim->exists (achieve = activity.isCompleted and dueTime = activity.endTime and sourceID = self.agentID and targetID = senderID)
Delete commitment	-	**not**(self.stitCommitmentClaim->exists (achieve = activity.isCompleted and dueTime = activity.endTime and sourceID = self.agentID and targetID = senderID))
Delete allocations	-	activity.productionActivityType. discreteStateResource->forAll (**not**(capacityInterval->exists (productionActivity = activity)))
Allocate resources	-	activity.productionActivityType. discreteStateResource->forAll (capacityInterval->exists (ci : CapacityInterval | ci.productionActivity->includes(activity) and activity.capacityInterval = ci))
Delete production activity and commitment (activity : ProductionActivity)	ProductionActivity.allInstances->exists (a : ProductionActivity | a.isScheduled and activity = a)	**not**(self.stitCommitmentClaim->exists (achieve = activity.isCompleted and dueTime = activity.endTime and sourceID = self.agentID and targetID = senderID)) **and not**(ProductionActivity.allInstances->exists (a : ProductionActivity | a = activity))
Delete commitment	-	**not**(self.stitCommitmentClaim->exists (achieve = activity.isCompleted and dueTime = activity.endTime and sourceID = self.agentID and targetID = senderID))
Delete allocations	-	activity.productionActivityType. discreteStateResource->forAll (**not**(capacityInterval->exists (productionActivity = activity)))
Delete production activity	-	**not**(ProductionActivity.allInstances->exists (a : ProductionActivity | a = activity))

Table 4-27 (continued). Activities of the case study of the ceramic factory with their preconditions and goals.

Register the start of the production activity (activity : productionActivity)	ProductionActivity.allInstances->exists (a : ProductionActivity | a.isScheduled and activity = a)	activity.isInProcess
Register the end of the production activity (activity : ProductionActivity)	ProductionActivity.allInstances->exists (a : ProductionActivity | a.isInProcess and activity = a)	activity.isCompleted **and** **not**(self.stitCommitmentClaim->exists (achieve = activity.isCompleted and dueTime = activity.endTime and sourceID = self.agentID and targetID = productionDepartment.agentID))
Resolve capacity conflict (resource : DiscreteState Resource)	DiscreteStateResource. allInstances->exists (r : DiscreteStateResource | r.hasCapacityConflict and resource = r)	**not** resource.hasCapacityConflict
Delete commitments and resource allocations	-	**not** resource.hasCapacityConflict and resource.productionActivityType->forAll (productionActivity->forAll(a : ProductionActivity | **not** a.isScheduled and **not** self.stitCommitmentClaim->exists (achieve = a.isCompleted and dueTime = a.endTime and sourceID = self.agentID and targetID = productionDepartment.agentID)))
Reschedule production activities	-	resource.productionActivityType->forAll (productionActivity->select(**not** isInProcess **and** not isCompleted)->forAll (a : ProductionActivity | a.isScheduled and self.stitCommitmentClaim->exists (achieve = a.isCompleted and dueTime = a.endTime and sourceID = self.agentID and targetID = productionDepartment.agentID))

4.1.5.5. Behaviour Modelling

At the step of behaviour modelling, function and motivation models of business processes by the goal-based use cases presented in Tables 4-1 - 4-26 are transformed into behaviour models by following the guidelines provided in section 3.8.5.2.

According to guideline 1, the <*time or sequence factor*> "*The invoice is authorized by the sales manager*" of step 3 of use case 5 ("Have the product set delivered") is represented as the non-communicative action event type authorizeInvoice connected to reaction rule R14.

Behaviour modelling based on the same guideline also enables to represent the behavioural pattern "Deferred choice" which was described in section 3.8.5.3. For example, the <*time or sequence factor*> components "*The acceptance of the proposal is received from the customer*" and "*The rejection of the proposal is received from the customer*" of steps 2 and 2a of use case 3 ("Make a proposal and process the reply") are modelled in Appendix F as the respective communicative action event types accept-proposal and reject-proposal which are connected to reaction rules R6 and R7, respectively. Action events of both types are triggered by an external agent of the type Customer. Analogously, the three alternative steps of use case 12 ("Follow the production activities") are modelled in Appendix F as reaction rules R30, R31, and R32 which are triggered by an external agent of the type ResourceUnit.

According to guideline 2 presented in section 3.8.5.2, the <*condition*> component "*Until the production order is completed*" of step 2 of use case 11 ("Complete the production order") is modelled in Appendix F as reaction rule R36 with the corresponding precondition that invokes an activity of the type "Follow production activities" if the instance of ProductionOrder under processing does *not* have the status isCompleted. This results in a "Repeat-Until" behavioural pattern that was described in section 3.8.5.3.

Based on the same guideline, the <*condition*> component of a use case step expressing the repetition for a number of instances of some informational entity type is represented as the corresponding reaction rule along with its precondition. For example, the <*condition*> components included by use cases 8 ("Instantiate the production plan") and 9 ("Schedule the production order") are turned in Appendix F into reaction rules R21 and R23, respectively, which form the corresponding "For-Each" loop patterns described in section 3.8.5.3.

According to guideline 4 provided in section 3.8.5.2, a precondition arrow of either reaction rule mentioned above is augmented by an OCL equation limiting the set of instances of the informational entity type for which the "For-Each" loop is performed. The OCL expression {productType = order.productType} attached to the precondition arrow of reaction rule R21 specifies that the action part of the rule (starting an activity of the type "Instantiate production activity") is repeated for each instance of ProductionActivityType that is associated with the same instance of ProductType as the instance of ProductionOrder referred to by the value of the input parameter order. In the same way, the OCL expression {productSet.productionOrder = order and isNextActivity(order)} attached to the precondition arrow of reaction rule R23 specifies that the action part of the rule (starting an activity of the type "Have production activity scheduled") is repeated for each instance of ProductionActivity that is associated with the instance of ProductSet corresponding to the given ProductionOrder. At each step of the loop, the intensional predicate isNextActivity(ProductionOrder) evaluated for the current instance of the object type ProductionActivity determines the next production activity for which an activity of the type "Have production activity scheduled" is started. The intensional predicate mentioned, which is defined in Appendix D, thus determines the order of scheduling production activities.

As another example of an augmented condition arrow, the OCL expression {orderID = ?String} attached in Appendix F to the precondition arrow of reaction rule R11 determines that the action part of the rule is performed (i.e., an activity of the type "Manage product delivery" is started) only for the instance of ProductionOrder the value of whose attribute orderID is equal to the value of the corresponding data item ?String included by the triggering communicative action event. An example of a more complicated precondition is {discreteStateResource = resource and isSchedulable(activity)} of reaction rule R46 stating that the instance of CapacityInterval to be assigned to the ProductionActivity by the rule is the one that (1) belongs to the given instance of DiscreteStateResource, which is referred to by the value of the input parameter resource, and (2) can be scheduled to the given instance of ProductionActivity. The schedulability of a capacity interval is determined by evaluating the intensional predicate isSchedulable (ProductionActivity) of the object type CapacityInterval. This predicate is defined in Appendix D.

Following guideline 3 provided in section 3.8.5.2, in Appendix F the symbols for the communicative action event types request achieve(isCompleted(ProductionOrder(?String))) and inform(isCompleted(?ProductionOrder)) are connected to the respective reaction rules R8 and R9 which are included by the elementary activity types "Request completion" and "Register completion", respectively. In the same way, the symbol for the communicative action event type propose provideProductSet(?ProductionOrder) is connected to reaction rule R5 which is included by the elementary activity type "Authorize and send proposal". The mental effect arrow originating in the same activity type is also connected to reaction rule R5. Analogously, the symbol for the non-communicative action event type provideProductSet(?ProductionOrder) and the arrow defining the accompanying mental effect are connected to reaction rule R12 which is included by the elementary activity type "Deliver product set". Based on the same guideline, the arrow denoting the activity starting action type START ACTIVITY "Manage scheduling and completion" is connected to reaction rule R2. The activity type "Manage scheduling and completion" is added because it enables to process the production order created. It does not have a counterpart in the goal-based use cases presented in Tables 4-1 – 4-26.

As was described in section 3.6.5, a mental effect arrow of a reaction rule may be augmented by a logical OCL expression that (re)defines the mental effect of the rule. For example, the OCL expression {productCode = code and quantity = quantity} attached to the mental effect arrow of reaction rule R2 in Appendix F determines that the values of the attributes productCode and quantity of the instance of ProductionOrder to be created by the rule should be equal to the values of the respective input parameters code and quant of the enclosing activity of the type "Manage production order".

According to guideline 4 provided in section 3.8.5.2, also a precondition arrow may be augmented by an OCL expression. For example, in addition to specifying the creation of an instance of ProductionOrder, the augmented precondition arrow and the mental effect arrows of reaction rule R19 in Appendix F jointly determine the association to be created by the rule between the instance of ProductionOrder created by the rule and the instance of ProductType that is identified by the value of its attribute productCode which must be equal to the value of the production order's attribute of the same name. The precondition and mental effects of reaction rule R22 determine that the instance of ProductionActivity created has the status isUnscheduled; its attribute typeName has the same value as the attribute activityName of the instance of ProductionActivityType that is referred to by the value of the input parameter type, and the ProductionActivity is associated with the instance of ProductionActivityType mentioned. In addition, the same reaction rule determines that there should be the two-way association between the instance of ProductionActivity created and the instance of ProductSet associated with the given instance of ProductionOrder which is referred to by the value of the input parameter order.

Analogously, the precondition and mental effect of reaction rule R46 determine that there should be the bidirectional association between the instance of CapacityInterval within the scope of the rule, referred to by the rule's internal variable CapacityInterval, and the instance of ProductionActivity to be scheduled, referenced by the value of the input parameter activity. In the same way, according to the mental effect expression of rule R47, the values of the attributes startTime and endTime of the instance of ProductionActivity to be scheduled by the rule are to be equal to respectively the starting time of the earliest and ending time of the latest CapacityInterval that is allocated to the ProductionActivity.

Another complicated example of augmentation is the OCL equation {earliestStartTime = getEarliestStartTime(order)} which is attached to the mental effect arrows of reaction rules R23, R33, and R37. This mental effect expression determines that the value of the attribute earliestStartTime of the ProductionActivity to be scheduled should be equal to the earliest start time of the production activity that is calculated by using the OCL definition getEarliestStartTime(ProductionOrder) presented in Appendix D. According to the definition mentioned, the earliest start time of the production activity is equal to either the release date of the production order (in case of its first production activity) or to the end time of the production order's last ProductionActivity scheduled so far increased by the shortest possible interval between an activity of that type and an activity of the given type.

4.1.5.6. Simulation of the Models on the JADE Agent Platform

In [Wagner03b] it has been shown that, with some minor extensions, AOR models can be used for a certain form of agent-based discrete event simulation, called *Agent-Object-Relationship Simulation* (AORS). An AORS system includes an environment simulator that is responsible to simulate exogenous events and the causality laws of the physical environment. Such a simulator can also be created for the case study of the ceramic factory. When applied jointly with the principles of creating executable process models worked out in this thesis, AORS enables to create powerful simulation environments. For simulating the business/manufacturing processes described by the models of the ceramic factory, the models described in section 4.1.5.1 to 4.1.5.5 were implemented in the Java language [JAVA] on the JADE agent platform in the way described in section 3.8.6. We will next treat by views of agent-oriented modelling how the executable JADE-based models corresponding to the executable models of the ceramic factory expressed by means of the extended AORML were created.

Organizational and Informational View

The types of institutional agents represented in the organization model of the ceramic factory shown in Figure 4-1 were implemented as the corresponding subclasses of the JADE's object class jade.core.Agent. This way, the Java classes CustomerAgent, SalesDepartmentAgent, ProductionDepartmentAgent, and ResourceUnitAgent were obtained. Their instances form the agents of the simulation environment that has been worked out as a part of the case study of the ceramic factory.

The informational entity types of the problem domain's information model represented in the agent diagram of Figure 4-3 were turned into the corresponding Java classes ProductionOrder, ProductSet, ProductType, Invoice, ProductionActivity, ProductionActivityType, Resource, Capacity, CapacityInterval, and PrecedenceInterval. Sets of their instances form the VKB's of the corresponding agents. In the example presented below, instantiations of the object classes ProductionOrder, ProductSet, ProductType, ProductionActivity, ProductionActivityType, PrecedenceInterval, and Resource form the VKB of the corresponding agent instance of the class ProductionDepartmentAgent:

```
public class ProductionDepartmentAgent extends jade.core.Agent {

   /** Virtual Knowledge Base */
   public HashMap productionOrder = new HashMap();
   public HashMap productionActivity = new HashMap();
   public HashMap productionActivityType = new HashMap();
   public HashMap resource = new HashMap();
   public HashMap productType = new HashMap();
   public HashSet precedenceInterval = new HashSet();

   /** Information about ontology */
   private Codec codec = new SLCodec();
   private Ontology ontology = TEKTOntology.getInstance();

   /** Earliest start time of the current production activity */
   public Date earliestStartTime;

   /** Latest end time of the current production activity */
   public Date latestEndTime;

   ...
```

}Out of the object classes mentioned, ProductType, ProductionActivityType, and PrecedenceInterval are pre-initialized within the agent instance.

In compliance with the agent diagram of Figure 4-3, the class Resource has the subclasses ReusableResource and DiscreteStateResource. The latter has been further divided into the subclasses UnitCapacityResource and BatchCapacityResource, and the class CapacityInterval has accordingly the subclasses UnitCapacityInterval and BatchCapacityInterval. The instances of UnitCapacityInterval and BatchCapacityInterval are included by the instances of UnitCapacityResource and BatchCapacityResource, respectively. The representation of the agent type Customer within the instance of SalesDepartmentAgent has been implemented as the Java object class of the same name. The attributes of an informational entity type modelled in Figure 4-3 have been implemented as attributes of the corresponding Java class, while status and intensional predicates of an informational entity type have been implemented as functions attached to the corresponding Java class. For example, the intensional predicate isSchedulable(ProductionActivity) of the object type CapacityInterval in Figure 4-3 has been implemented as the Java function with the signature public boolean isSchedulable(ProductionActivity activity). One-to-one associations between informational entity types have been implemented as references between

instances of the corresponding object classes, like is shown in the example below. One-to-many associations have been implemented as ordered Java collections containing references to the appropriate object instances. In the example below, a reference within an instance of ProductSet to the ordered sequence of instances of ProductionActivity, which define the set of processing steps required to produce the ProductSet, has been implemented by means of a Java collection of the type ArrayList. An association class like PrecedenceInterval has been implemented as an ordinary Java class where any instance includes the references to the instances of the Java object classes that correspond to the informational entity types the association applies to.

The Java object class corresponding to the object type ProductSet of the informational view looks like as follows:

```
public class ProductSet extends Object implements Concept {

   /** References */
   private ProductionOrder productionOrder;
   private ProductType productType;

   /** Ordered list of activities of the product */
   private ArrayList productionActivity = new ArrayList();

   /** Creates new ProductSet for the given ProductionOrder */
   public ProductSet(ProductionOrder order) {
      productionOrder = order;
      productType = productionOrder.getProductType();
   }

...
}
```

Interactional View

Since the agents of the simulation system must understand messages received from each other, the JADE-based ontology of the problem domain was created. This ontology corresponds to the union of shared object types and action event types that are defined by the extended AORML models of the informational and interactional views. The ontology of the ceramic factory extends the basic ontology that is defined in the JADE development library (jade.content.onto.Ontology).

According to the principles laid out in section 3.8.6.2, the concept schemas of the CeramicFactoryOntology were created for all Java classes that implement the informational entity types of the information model. Each schema added to the ontology was associated with the corresponding Java class. For example, the schema for the ProductionActivity concept was associated with the class ProductionActivity. When using the ontology, expressions indicating production activities are instances of the ProductionActivity class. All Java classes corresponding to concept schemas implement the jade.content.Concept interface.

The non-communicative action event types modelled in Figure 4-4 were mapped into the corresponding agent action schemas defining the structure of the agent actions relevant to the domain addressed. Since each schema added to the ontology must be associated with a Java class [JADE], the Java classes provideProductSet, payInvoice, and achieve implementing the jade.content.AgentAction interface were created. This was followed by the creation of the corresponding agent action schemas of the ontology. While an agent action of the type provideProductSet or payInvoice is used with a concept or primitive, an action of the achieve type is always accompanied by a predicate. The CeramicFactoryOntology thus includes the predicate schemas corresponding to the status predicates isScheduled, isInProcess, and isCompleted. For these schemas were also created the corresponding Java objects implementing the jade.content.Predicate interface.

Agent messages modelled in Figure 4-4 either directly contain concepts like ?ProductionOrder or just reference them by using a concept's identifier attribute like ProductionOrder(?String). In the first case, the representation of the corresponding concept within an agent message includes as primitives the values of some or all attributes of the corresponding Java object. In the second case, the representation of the concept consists of just the value of the object's identifier attribute.

Functional and Behavioural Views

The behavioural constructs represented in the activity diagrams of the case study in Appendix F were rather straightforwardly mapped to the corresponding constructs of the JADE framework. Firstly, as is described in section 3.8.6.3, an object class MessageHandler for processing incoming agent messages and starting first-level activities was defined for each agent type and instantiated for each agent

instance of the simulation environment. For example, the instance of MessageHandler of the agent SalesDepartment incorporates the behaviours prescribed by reaction rules R1, R11, and R17. Analogously, the instance of MessageHandler created for the agent ProductionDepartment processes incoming agent messages and starts first-level activities as is determined by reaction rules R19, R27, and R28. For processing input from a human agent through a GUI, the object class InputHandler was defined for each agent type and instantiated for each agent instance of the simulation environment. For example, the InputHandler of a ResourceUnit incorporates the actions specified by reaction rules R58, R60, and R62.

Next, the Java classes corresponding to the activity types of the behaviour model were created. For example, according to the guidelines provided in section 3.8.6.3, the activity type "Manage production order" was implemented as the object class Manage_production_order which extends the JADE's class SequentialBehaviour. An instance of the class Manage_production_order executes in sequential order its sub-behaviours corresponding to the instances of the activity types "Create production order" and "Manage scheduling and completion". The first of these activity types has been implemented as the subclass Create_production_order of OneShotBehaviour, while the second activity type has been represented by the class Manage_scheduling_and_completion extending the JADE's class SequentialBehaviour. Passing the value of the input parameter of the type ProductionOrder from the preceding activity of the first type to the following activity of the second type has been implemented through the use of the corresponding public variable order of their "father" activity of the class Manage_production_order. The activity type "Manage scheduling and completion", which has been implemented by the behaviour class of the same name, in turn consists of the subactivity types "Have production order scheduled", "Manage proposal", and "Manage completion". All of them have been implemented as the corresponding subclasses of jade.core.behaviours.SequentialBehaviour. The class Manage_proposal, which is presented as an example below, implements the behavioural pattern "Deferred choice" by creating a subclass of the JADE's class jade.core.behaviours.ParallelBehaviour whose instance executes concurrently as many children behaviours of the class jade.core.behaviours.ReceiverBehaviour as there are options for different messages received by the agent. As soon as any of the sub-behaviours is done, i.e. the corresponding message has been received, the parallel behaviour terminates. Messages of two kinds – "accept-proposal" and "reject-proposal" – can be received in the "Deferred choice" construct presented below:

```
class Manage_proposal extends SequentialBehaviour {

   /** Placeholder for the received message */
   private ACLMessage receivedMessage;

   /** Production order under processing */
   private ProductionOrder productionOrder;

   /** Constructor of the behaviour */
   public Manage_proposal(ProductionOrder order, ACLMessage msg) {
      super(agent);
      productionOrder = order;
      receivedMessage = msg;
   }

   public void action() {
      addSubBehaviour(new Authorize_and_send_proposal(order, msg));
      myAgent.addBehaviour(new TemporaryParallelBehaviour (WHEN_ANY));
   }
}

class TemporaryParallelBehaviour extends ParallelBehaviour {

   public void action() {
      MessageTemplate mt1 =
        MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
      MessageTemplate mt2 =
        MessageTemplate.MatchPerformative(ACLMessage.REJECT_PROPOSAL);

      // Add sub-behaviours
      addSubBehaviour(new ReceiverBehaviour1(myAgent, -1, mt1));
      addSubBehaviour(new ReceiverBehaviour2(myAgent, -1, mt2));
   }
}
```

With the help of the visual user interface of the simulation system created for the ceramic factory, the user is, for example, able to describe an instance of ProductionOrder by evaluating the attributes releaseDate, dueDate, quantity, and specifying a reference to the corresponding ProductType. After that, the system computes the preliminary schedule and presents it to the user. We also plan to introduce the "performing" of the schedule by the agent-based simulation system and reactions to capacity and time conflicts resulting from it, as described in [Smith95]. Later on, the system can be connected to the actual production environment where it would be used for the creation and dynamical adjustment of production schedules.

4.2. THE CASE STUDY OF ADVERTISING

Since the paradigm of agent-orientation promotes *autonomous action and decision-making*, it is highly relevant for modelling and implementing business processes involving different enterprises with their respective information systems. Considering this, we have used the methodology proposed by us for modelling inter-enterprise business processes types of the advertising domain. This can be seen as the first step towards automating these processes.

4.2.1. Overview of the Domain

According to [Antikainen01], advertising is an important source of revenue for newspapers. Classified ads are migrating to the Internet and in many countries newspapers have lost their market share to other media. To maintain and improve their position, newspapers need to be active and start developing electronic advertising processes in co-operation with their customers. The newspaper industry should make newspaper advertising as easy as possible for its customers to maintain and improve its competitive position. Also, it is essential that newspapers develop the advertising processes in order to reduce costs and improve productivity. Process development requires co-operation between all parties that are involved in the advertising process. This way costs can be reduced not only in newspapers, but also in the whole advertising chain making newspapers a more attractive advertising media.

The analysis of current processes performed in [Antikainen01] revealed that a characteristic feature in newspaper advertising processes is the large number of parties. Publishing an ad in a newspaper typically needs contributions from an advertiser, media agency, ad agency, repro house, and courier service provider. The number of parties' means that many of people are involved, which makes the communication costs high. The need for communication is increased also by the number of changes in ad space reservations and ad orders. The hassle in advertising processes affects the whole chain and results in a considerable number of errors in invoicing. For these reasons, it is very important that the newspaper duly has all the information that is needed to process and place an ad. This is also for the benefit of the advertising customer [Antikainen01] .

Typically, the ad order process in European countries is paper based. Newspapers receive ad space reservations by phone and ad orders by fax or mail. Ad orders and the corresponding artwork are manually connected to each other. Copies of ad orders are used to control ad production and ad placement in newspapers. After the publication, ad sizes are measured and tearsheets may be cut and attached manually to the invoice. To lower cost, these manual processes need to be developed and transferred into electronic processes [Antikainen01] .

Electronic artwork delivery from computer to computer is already common practice in many countries. But unfortunately many newspapers support too many ways to receive electronic artwork. The worst is to accept email attachments, which may never arrive in the newspapers. Also, the information on the artwork may be incomplete making it impossible to place and process the artwork without further inquiries.Many electronic artwork delivery services help the newspapers to receive artwork files that are according the newspaper's requirements by preflighting them and returning those that do not pass the preflight check and even advising in correcting them. Some electronic artwork delivery services have introduced or are planning to introduce a possibility to deliver electronic ad orders as well. In these services the material transfer form is extracted from the electronic ad order. These services combine the electronic ad order and electronic artwork automatically – a time consuming task that is usually done manually in newspapers [Antikainen01].

4.2.2. Goals of the Case Study

As the overview presented in section 4.2.1 revealed, at present there is not much automation in the advertising domain. In our opinion, *the first step towards introducing more automation into the domain is developing proper modelling notations and methodologies which enable to integrate seamlessly the modelling of business processes and information they make use of*. The business processes modelled should cover the whole advertising campaign. A possible next step would be further developing of web auctions for selling surplus advertising space mentioned in [Antikainen01]. According to the same source, there is usually not enough time to sell surplus ad space in web auctions. An automation solution possibly making use of software agents would be in the position of making advertising processes to work almost in real time. Such a solution could be used for selling premium ad space in addition to the surplus one. This would result in the increase of the speed of advertising business processes which would be positively reflected by the participants' cash flows.

4.2.3. Analysis with Goal-Based Use Cases

The use cases in Tables 4-28 – 4-52 describe the inter-organizational business process types of advertising with different types of actors, sketched as a part of the analysis step, in focus. In Table 4-28, use case 1 "Carry out the advertising campaign", which has the media agency in focus, is presented. This use case is triggered by receiving from an advertiser a request to perform the advertising campaign specified by the campaign order of a predefined and agreed between the actors form. The goal of the first use case, "expecting all the ad orders corresponding to the advertising campaign to be created and submitted", is given in its context in an informal way. It is semi-formalized in section 4.2.4.3 at the modelling phase of design. The use case is modelled from the perspective of the advertiser with the media agency in focus (*scope*) which means that the goal of the use case is the so-called *user goal*, the goal of the actor (i.e., the advertiser) trying to get work (*primary task*) done. Since the use case "Carry out the advertising campaign" is triggered by the advertiser, the advertiser is called the *primary actor* of the use case. The publication and artwork designer are termed *secondary actors* because they are the ones from which the actor in focus, the media agency, needs assistance to satisfy the user goal internalized by it. The artwork designer, in turn, has the artwork producer as its secondary actor. Other primary tasks, i.e. use cases that are triggered by primary actors, are use cases 8, 10, 11, 12, 14, 15, 18, 23, and 25 below.

Use case 1 includes as *subfunctions* use cases 2, 3, 6, and 7. As we learned in section 3.7.1, the goal of a *subfunction*, which is a subgoal of some user goal, is attached to the actor in focus. For example, the goal "expecting ad space to be reserved in the publications according to the campaign order" of the subfunction "Have ad space reserved in the publications" (use case 3), which is a subgoal of the user goal "expecting all the ad orders corresponding to the advertising campaign to be created and submitted", is attached to the media agency. The second-level subfunction of the use case mentioned, use case 5 "Evaluate the ad space reservation proposal" includes the main scenario for the case the ad space reservation proposal is accepted by the internal actor 'media agency secretary' and the *extension scenario* for the opposite case.

A special group of subfunctions are subfunctions that are triggered by internal actors. In the example of advertising, to this group belong use cases 6, 7, 17, 19, and 24. For example, use case 6 is triggered by the media agency secretary, while use case 7 is triggered by the internal actor 'timer' of the media agency.

In Tables 4-28 to 4-52, the <*time or sequence factor*> and <*condition*> components of use case steps are distinguished by representing them in *italic*.

Table 4-28. Extended use case for the business process "Carry out the advertising campaign".

USE CASE 1	Carry out the advertising campaign.	
Goal in Context	An advertiser expects all the ad orders corresponding to the advertising campaign to be created and submitted.	
Scope & Level	Media agency, primary task.	
Preconditions		
Success End Condition	All the ad orders corresponding to the advertising campaign have been created and submitted.	
Primary Actor **Secondary Actors**	Advertiser. Publication, artwork designer.	
Trigger	A request by the advertiser to perform the advertising campaign according to the campaign order.	
DESCRIPTION	**Step**	**Action**
	1	*The advertising campaign is authorized by the media agency secretary*: the media agency sends to the advertiser an agreement to perform the advertising campaign and commits towards the advertiser to perform the campaign.
	2	*The campaign order includes an artwork description*: the media agency orders the artwork to be designed (**Use Case 2**).
	3	The media agency has ad space reserved in the publications according to the campaign order (**Use Case 3**).
	4	*A request to update the ad space reservation by the media agency secretary is received*: have the ad space reservation updated (**Use Case 6**).
	5	*A request by the timer to request printing of the ad is received*: request printing of the ad (**Use Case 7**).

Table 4-29. Extended use case for the business process "Request artwork design".

USE CASE 2	Request artwork design.	
Goal in Context	The media agency expects the artwork to be designed according to the campaign order.	
Scope & Level	Artwork designer, subfunction.	
Preconditions	The media agency has agreed and committed to perform the advertising campaign.	
Success End Condition	The artwork designer has agreed to design the artwork.	
Primary Actor **Secondary Actors**	Advertiser. Artwork designer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The media agency sends to the artwork designer a request to design the artwork.
	2	The media agency receives from the artwork designer an agreement to design the artwork.

Table 4-30. Extended use case for the business process "Have ad space reserved in the publications".

USE CASE 3	Have ad space reserved in the publications.	
Goal in Context	The media agency expects ad space to be reserved in the publications according to the campaign order.	
Scope & Level	Media agency, subfunction.	
Preconditions	The media agency has agreed and committed to perform the advertising campaign.	
Success End Condition	The ad space has been reserved in the publications according to the campaign order.	
Primary Actor **Secondary Actors**	Advertiser. Publication.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each ad insertion included by the campaign order*: have ad space reserved in the corresponding publication according to the ad insertion (**Use Case 4**).

Table 4-31. Extended use case for the business process "Have ad space reserved according to the ad insertion".

USE CASE 4	Have ad space reserved according to the ad insertion.	
Goal in Context	The media agency expects ad space to be reserved in the publication according to the ad insertion.	
Scope & Level	Media agency, subfunction.	
Preconditions	The media agency has agreed and committed to perform the advertising campaign.	
Success End Condition	The ad space has been reserved in the publication according to the ad insertion.	
Primary Actor **Secondary Actors**	Advertiser. Publication.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The media agency creates an ad space reservation request for the ad specified by the ad insertion.
	2	The media agency sends the ad space reservation request to the corresponding publication.
	3	*The media agency receives from the publication a proposal for ad space reservation*: the proposal is evaluated within the media agency (**Use Case 5**).
EXTENSIONS	**Step**	**Branching Action**
	3a	*The media agency receives from the publication a refusal to reserve ad space*: the ad space reservation request is deleted and the business process ends.

Table 4-32. Extended use case for the business process "Evaluate the ad space reservation proposal".

USE CASE 5	Evaluate the ad space reservation proposal.	
Goal in Context	The media agency expects the ad space reservation to be created.	
Scope & Level	Media agency, subfunction.	
Preconditions	The media agency has received from the publication a proposal for ad space reservation.	
Success End Condition	The ad space reservation has been created.	
Primary Actor **Secondary Actors**	Advertiser. Publication.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*The proposal is accepted by the media agency secretary*: the ad space reservation is created, the publication is informed about the acceptance, and the timer is requested to turn the ad space reservation into the ad order at the time specified in the ad order.
EXTENSIONS	**Step**	**Branching Action**
	1a	*The proposal is rejected by the media agency secretary*: the ad space reservation request is deleted and the business process ends.

Table 4-33. Extended use case for the business process "Have the ad space reservation updated".

USE CASE 6	Have the ad space reservation updated.	
Goal in Context	The media agency expects the ad space reservation to be updated.	
Scope & Level	Media agency, subfunction.	
Preconditions	The ad space reservation has been created.	
Success End Condition	The ad space reservation has been updated.	
Primary Actor **Secondary Actors**	Advertiser. Publication.	
Trigger	A request to update the ad space reservation by the media agency secretary.	
DESCRIPTION	**Step**	**Action**
	1	The media agency sends to the publication a request to update the ad space reservation.
	2	The media agency receives from the publication a confirmation on the update of the ad space reservation, and registers it.

Table 4-34. Extended use case for the business process "Request printing of the ad".

USE CASE 7	Request printing of the ad.	
Goal in Context	The media agency expects the ad space reservation to be turned into the ad order.	
Scope & Level	Media agency, subfunction.	
Preconditions	The ad space reservation has been created.	
Success End Condition	The ad space reservation has been turned into the ad order.	
Primary Actor **Secondary Actors**	Advertiser. Publication.	
Trigger	A request by the timer to request printing of the ad.	
DESCRIPTION	**Step**	**Action**
	1	The media agency sends to the publication a request to provide it with the printed ad corresponding to the ad space reservation.
	2	The media agency receives from the publication an agreement to provide it with the printed ad along with the corresponding confirmed ad order, and registers the ad order.

Table 4-35. Extended use case for the business process "Complete the advertising campaign".

USE CASE 8	Complete the advertising campaign.	
Goal in Context	The publication expects the printing of the ad to be registered.	
Scope & Level	Media agency, primary task.	
Preconditions	The ad order corresponding to the ad provided by the publication exists.	
Success End Condition	The printing of the ad has been registered.	
Primary Actor **Secondary Actors**	Publication. Advertiser.	
Trigger	Arrival of the printed ad from the publication.	
DESCRIPTION	**Step**	**Action**
	1	The media agency registers the printing of the ad.
	2	*The advertising campaign has been performed (ads described by all the ad orders corresponding to the campaign order have been printed)*: the media agency creates an invoice and sends it to the advertiser (**Use Case 9**).

Table 4-36. Extended use case for the business process "Create and send an invoice to the advertiser".

USE CASE 9	Create an invoice and send it to the advertiser.	
Goal in Context	The media agency expects an invoice corresponding to the advertising campaign performed to be created and sent to the advertiser.	
Scope & Level	Media agency, subfunction.	
Preconditions	The advertising campaign has been performed.	
Success End Condition	The media agency invoice has been created and sent to the advertiser.	
Primary Actor **Secondary Actors**	Publication. Advertiser.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The media agency creates the media agency invoice.
	2	*The media agency invoice is approved by the media agency secretary*: the media agency sends the invoice to the advertiser and creates the claim against the advertiser to pay for the advertising campaign according to the invoice.

Table 4-37. Extended use case for the business process "Register the payment by the advertiser".

USE CASE 10	Register the payment by the advertiser.	
Goal in Context	The advertiser expects the media agency to accept the payment for the advertising campaign.	
Scope & Level	Media agency, primary task.	
Preconditions	The media agency invoice has been created and sent to the advertiser.	
Success End Condition	The media agency has accepted the payment by the advertiser according to the media agency invoice.	
Primary Actor **Secondary Actors**	Advertiser.	
Trigger	Receiving of a payment by the advertiser.	
DESCRIPTION	**Step**	**Action**
	1	The media agency registers the payment and satisfies the claim against the advertiser to pay for the campaign according to the media agency invoice.

Table 4-38. Extended use case for the business process "Process the publication invoice".

USE CASE 11	Process the publication invoice.	
Goal in Context	The publication expects the media agency to pay for the ad according to the publication invoice.	
Scope & Level	Media agency, primary task.	
Preconditions	The ad corresponding to the publication invoice has been printed.	
Success End Condition	The media agency has agreed to pay for the ad according to the publication invoice.	
Primary Actor Secondary Actors	Publication.	
Trigger	A request by the publication to pay for the ad according to the publication invoice.	
DESCRIPTION	**Step**	**Action**
	1	*The invoice is accepted by the media agency secretary*: the media agency sends to the publication an agreement and commits to pay for the ad according to the publication invoice.
EXTENSIONS	**Step**	**Branching Action**
	1a	*The invoice is rejected by the media agency secretary*: the media agency sends to the publication a refusal to pay for the ad according to the publication invoice.

Table 4-39. Extended use case for the business process "Reserve ad space".

USE CASE 12	Reserve ad space.	
Goal in Context	The media agency expects ad space to be reserved for the ad according to the ad space reservation request.	
Scope & Level	Publication, primary task.	
Preconditions		
Success End Condition	The ad space reservation has been created.	
Primary Actor Secondary Actors	Media agency.	
Trigger	An ad space reservation request by the media agency.	
DESCRIPTION	**Step**	**Action**
	1	*There is sufficiently ad space or alternative ad space for the reservation request in question*: the publication sends to the media agency a proposal for ad space reservation and processes the reply by the media agency (**Use Case 13**).
EXTENSIONS	**Step**	**Branching Action**
	1a	*There is not enough ad space or alternative ad space*: the publication deletes the ad space reservation request and sends to the media agency a refusal to reserve ad space.

Table 4-40. Extended use case for the business process "Process the reply by the media agency".

USE CASE 13	Process the reply by the media agency.	
Goal in Context	The publication expects the proposal for ad space reservation to be accepted by the media agency.	
Scope & Level	Media agency, subfunction.	
Preconditions	There is sufficiently ad space or alternative ad space for the reservation request received by the publication.	
Success End Condition	The ad space reservation has been created.	
Primary Actor Secondary Actors	Media agency.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*The acceptance of the proposal for ad space reservation is received from the media agency*: the publication creates the ad space reservation.
EXTENSIONS	**Step**	**Branching Action**
	1a	*The rejection of the proposal for ad space reservation is received from the media agency*: the publication deletes the ad space reservation request and the business process ends.

Table 4-41. Extended use case for the business process "Update the ad space reservation".

USE CASE 14	Update the ad space reservation.	
Goal in Context	The media agency expects the ad space reservation to be updated.	
Scope & Level	Publication, primary task.	
Preconditions	The ad space reservation has been created.	
Success End Condition	The ad space reservation has been updated.	
Primary Actor Secondary Actors	Media agency.	
Trigger	A request by the media agency to update the ad space reservation.	
DESCRIPTION	**Step**	**Action**
	1	The publication updates the ad space reservation and sends the updated ad space reservation to the media agency.

Table 4-42. Extended use case for the business process "Have the ad printed".

USE CASE 15	Have the ad printed.	
Goal in Context	The media agency expects the ad to be printed.	
Scope & Level	Publication, primary task.	
Preconditions	The ad space reservation has been created.	
Success End Condition	The ad space reservation has been turned into the corresponding ad order.	
Primary Actor Secondary Actors	Media agency. Artwork producer.	
Trigger	A request by the media agency to provide it with the printed ad.	
DESCRIPTION	**Step**	**Action**
	1	The publication turns the ad space reservation into the ad order, sends the confirmed ad order to the media agency, and commits towards the media agency to provide it with the printed ad.
	2	*The order includes an artwork description*: receive the artwork (**Use Case 16**).
	3	*A signal by the publication secretary on printing of the ad is received*: the printing of the ad is registered (**Use Case 17**).

Table 4-43. Extended use case for the business process "Receive the artwork".

USE CASE 16	Receive the artwork.	
Goal in Context	The publication expects to receive the artwork and connect it to the ad order.	
Scope & Level	Publication, subfunction.	
Preconditions	The order includes an artwork description.	
Success End Condition	The artwork has been received and connected to the ad order.	
Primary Actor **Secondary Actors**	Media agency. Artwork producer.	
Trigger	The arrival of the artwork sent by the artwork producer.	
DESCRIPTION	**Step**	**Action**
	1	The artwork is connected to the ad order.

Table 4-44. Extended use case for the business process "Deal with the publication invoice".

USE CASE 17	Deal with the publication invoice.	
Goal in Context	The publication is willing to provide the media agency with the printed ad and expects to receive from the media agency an agreement to pay for the ad according to the publication invoice.	
Scope & Level	Publication, subfunction.	
Preconditions	The ad order has been created.	
Success End Condition	The ad has been sent to the media agency and the media agency has agreed to pay for the ad according to the publication invoice.	
Primary Actor **Secondary Actors**	Media agency.	
Trigger	Registration of the printing of the ad by the publication secretary.	
DESCRIPTION	**Step**	**Action**
	1	The publication provides the media agency with the printed ad.
	2	The publication secretary measures the size of the printed ad and updates the ad size in the ad order.
	3	The publication creates the publication invoice.
	4	*The publication invoice is approved and, optionally, changed by the publication secretary*: the publication sends the publication invoice to the media agency and creates the claim against the media agency to receive a payment for the ad according to the invoice.
	5	*An agreement to pay for the ad according to the publication invoice is received from the media agency*: the business process ends.
EXTENSIONS	**Step**	**Branching Action**
	5a	*A refusal to pay for the ad according to the publication invoice is received from the media agency*: go to step 4.

Table 4-45. Extended use case for the business process "Design the artwork".

USE CASE 18	Design the artwork.	
Goal in Context	The media agency expects the artwork specified by the campaign order to be designed.	
Scope & Level	Artwork designer, primary task.	
Preconditions	The campaign order includes the artwork description.	
Success End Condition	The artwork has been designed.	
Primary Actor **Secondary Actors**	Media agency. Advertiser.	
Trigger	A request by the media agency to design the artwork.	
DESCRIPTION	**Step**	**Action**
	1	*The artwork design request is approved by the artist*: the artwork designer commits to design the artwork and sends a confirmation to the media agency.
	2	*A signal on completing the artwork design by the artist is received*: register the designing of the artwork (**Use Case 19**).

Table 4-46. Extended use case for the business process "Register the designing of the artwork".

USE CASE 19	Register the designing of the artwork.	
Goal in Context	The artwork designer expects the designing of the artwork to be registered.	
Scope & Level	Artwork designer, subfunction.	
Preconditions	A request to design the artwork has been received by the artwork designer and approved by the artist	
Success End Condition	The artwork has been designed and the designing has been registered.	
Primary Actor **Secondary Actors**	Media agency. Advertiser.	
Trigger	A signal on completing the artwork design by the artist.	
DESCRIPTION	**Step**	**Action**
	1	The artwork designer stores the artwork design.
	2	The artwork designer sends a proof to the advertiser.
	3	*An acceptance of the proof is received from the advertiser*: the artwork designer satisfies the commitment towards the media agency to design the artwork and has the artwork produced (**Use Case 20**).
EXTENSIONS	**Step**	**Branching Action**
	3a	*A rejection of the proof is received from the advertiser*: the subfunction ends.

Table 4-47. Extended use case for the business process "Have the artwork produced".

USE CASE 20	Have the artwork produced.	
Goal in Context	The artwork designer expects the artwork to be produced.	
Scope & Level	Artwork designer, subfunction.	
Preconditions	The artwork has been designed.	
Success End Condition	The artwork has been produced.	
Primary Actor **Secondary Actors**	Media agency. Artwork producer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The artwork designer creates the artwork production order.
	2	The artwork designer sends to the artwork producer a request to produce the artwork.
	3	The artwork designer receives from the artwork producer an agreement to produce the artwork.
	4	The artwork designer receives from the artwork producer a proof of the artwork.
	5	*The proof is accepted by the artist*: the artwork designer informs the artwork producer about the acceptance of the proof and has the artwork distributed (**Use Case 21**).
EXTENSIONS	**Step**	**Branching Action**
	5a	*The proof is rejected by the artist*: the artwork designer informs the artwork producer about the rejection of the proof.

Table 4-48. Extended use case for the business process "Have the artwork distributed".

USE CASE 21	Have the artwork distributed.	
Goal in Context	The artwork designer expects the artwork to be sent to the publications included by the campaign order.	
Scope & Level	Artwork designer, subfunction.	
Preconditions	The artwork has been produced.	
Success End Condition	The artwork producer has been requested to send the artwork to the publications included by the campaign order.	
Primary Actor **Secondary Actors**	Media agency. Artwork producer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*For each publication included by the campaign order*: request the artwork producer to provide the publication with the artwork (**Use Case 22**).

Table 4-49. Extended use case for the business process "Request distribution".

USE CASE 22	Request distribution.	
Goal in Context	The artwork designer expects the artwork provider to accept the request to provide the publication with the artwork.	
Scope & Level	Artwork designer, subfunction.	
Preconditions	The artwork has been produced.	
Success End Condition	The artwork producer has agreed to provide the publication with the artwork.	
Primary Actor **Secondary Actors**	Media agency. Artwork producer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	The artwork designer sends to the artwork producer a request to provide the given publication with the artwork.
	2	The artwork designer receives from the artwork producer an agreement to provide the given publication with the artwork.

Table 4-50. Extended use case for the business process "Produce the artwork".

USE CASE 23	Produce the artwork.	
Goal in Context	The artwork designer expects the artwork specified by the artwork production order to be produced.	
Scope & Level	Artwork producer, primary task.	
Preconditions	The artwork production order including the artwork design has been created.	
Success End Condition	The artwork has been produced.	
Primary Actor **Secondary Actors**	Artwork designer. Publication.	
Trigger	A request by the artwork designer to produce the artwork specified by the artwork production order.	
DESCRIPTION	**Step**	**Action**
	1	*The artwork production request is approved by the production manager*: the artwork producer commits to produce the artwork according to the artwork production order and sends a confirmation to the artwork designer.
	2	*A signal on completing the artwork production by the production manager is received*: register the producing of the artwork (**Use Case 24**).

Table 4-51. Extended use case for the business process "Register the producing of the artwork".

USE CASE 24	Register the producing of the artwork.	
Goal in Context	The production manager expects the producing of the artwork to be registered.	
Scope & Level	Artwork producer, subfunction.	
Preconditions	The request to produce the artwork according to the artwork production order has been received by the artwork producer and approved by the artist.	
Success End Condition	The artwork has been produced and the producing has been registered.	
Primary Actor **Secondary Actors**	Media agency. Publication.	
Trigger	A signal on completing the artwork production by the production manager.	
DESCRIPTION	**Step**	**Action**
	1	The artwork producer stores the artwork.
	2	The artwork producer sends a proof to the artwork designer.
	3	*An acceptance of the proof is received from the atwork designer*: the artwork producer satisfies the commitment towards the artwork designer to produce the artwork and registers the acceptance.
EXTENSIONS	**Step**	**Branching Action**
	3a	*A rejection of the proof is received from the advertiser*: the subfunction is ended.

Table 4-52. Extended use case for the business process "Provide the publication with the artwork".

USE CASE 25	Provide the publication with the artwork.	
Goal in Context	The artwork designer expects the publication to be provided with the artwork.	
Scope & Level	Artwork producer, primary task.	
Preconditions	The artwork has been produced.	
Success End Condition	The publication has been provided with the artwork.	
Primary Actor **Secondary Actors**	Artwork designer. Publication.	
Trigger	A request by the artwork designer to provide the publication specified by the artwork connected to the artwork production order specified.	
DESCRIPTION	**Step**	**Action**
	1	The artwork producer commits to provide the publication with the artwork and sends a confirmation to the artwork designer.
	2	The artwork producer provides the publication with the artwork.

4.2.4. Design By Extended AOR Modelling

4.2.4.1. Organization and Information Modelling

Since in an inter-organizational setting every agent must have a knowledge of all the other agents and about objects of the problem domain, it is not reasonable or even always possible to separate organization and information models from each other. Therefore the organization and information models of the case study of advertising, which have been created based on [Antikainen01], are both represented in the agent diagram of Figure 4-5. The organization model of the advertising domain depicted in Figure 4-5 consists of the agent types Advertiser, MediaAgency, Publication, ArtworkDesigner, and ArtworkProducer. All of them are subtypes of the agent type Organization which is not shown in Figure 4-5. Any company can be an Advertiser. The agent types MediaAgency and Publication are subclasses of the institutional roles BuyerOfAdSpace and SellerOfAdSpace, respectively. The agent type MediaAgency includes the human agent type MediaAgencySecretary and the agent type Timer, both with just one instance. The agent type Publication consists of the human agent type ChiefEditor, which has exactly one instance, and of the following subtypes of OrganizationUnit: PrintingPlant, PrepressProduction, EditorialDepartment, and AdvertisingDepartment. For the latter is modelled the internal human agent type PublicationSecretary with one instance. Naturally, there is a control (isSubordinateTo) relationship between the human agent types PublicationSecretary and ChiefEditor. Since inter-organizational business processes are based on agreements between parties, there are also dependency (providesResourceTo) and benevolence (isBenevolentTo) relationships between the agent types shown in Figure 4-5. The agent types ArtworkDesigner and ArtworkProducer include the respective internal human agent types Artist and ProductionManager. All the agent types of the organization model of advertising define the identifier attribute agentID of the type String. In Figure 4-5, this attribute is shown for Publication.

The information model of the advertising domain includes the shared object types CampaignOrder and AdOrder. The instances of CampaignOrder are exchanged by agents of the types Advertiser, ArtworkDesigner, and MediaAgency, while instances of AdOrder are passed between agents of the types MediaAgency and Publication. A CampaignOrder can have the status isPreliminary, isArtworkDesigned, or isPerformed. An instance of AdOrder always has one of the following statuses: isPreliminary, isReserved, isRejected, isUpdated, isConfirmed, and isPrinted. The status predicate isPerformed of the object type CampaignOrder is defined as follows by using the extended OCL:

```
context CampaignOrder inv:
self.isPerformed IF self.adInsertion->forAll(adOrder->exists(isPrinted or isRejected))
```

An instance of CampaignOrder is associated with one or more instances of AdInsertion describing the particular ad to be published in the particular issue of the publication whose representation is included by the AdInsertion. A CampaignOrder also includes the object instance :AdDescription, which, in turn, consists of the internal object :AdSize, and, optionally, the object instance :ArtworkDescription. As Figure 4-5 shows, each :ArtworkDescription is associated with the instance of ArtworkDesign that it "describes". An instance of ArtworkDesign, in turn, is included by an object of the type ArtworkProductionOrder. The instances of the latter are exchanged by agents of the types ArtworkDesigner and ArtworkProducer. An ArtworkProductionOrder is characterized by the attribute of the type dueDate and the status predicates isPreliminary and isProduced. For an ArtworkProducer, an instance of ArtworkDesign determines the corresponding instance of Artwork which is consumed by agents of the type Publication. Both ArtworkDesign and Artwork, which are represented in a digital form, have the status predicate isOK which is used in the business processes of proof evaluation. The attributes of :AdDescription and :ArtworkDescription are not shown in Figure 4-5. The status predicates isArtworkDesigned of CampaignOrder and isProduced of ArtworkProductionOrder are defined in the following way:

```
context CampaignOrder inv:
self.isArtworkDesigned IF ArtworkDesign.allInstances->exists
(d : ArtworkDesign | d.artworkDescription = self.adDescription.artworkDescription and
self.adDescription.artworkDescription.artworkDesign = d and d.isOK)

context ArtworkProductionOrder inv:
self.isProduced IF Artwork.allInstances->exists(aw : Artwork |
aw.artworkDesign = self.artworkDesign and self.artworkDesign.artwork = aw and aw.isOK)
```

The agent type Publication includes the private object type Issue with the status predicate isPublished and intensional predicates hasAdSpaceFor and hasAlternativeAdSpaceFor. Either intensional predicate determines on the basis of the value of the derived attribute availableArea of the corresponding instance

of Issue whether there is enough ad space for the ad described by the given AdOrder. The intensional predicate hasAlternativeAdSpace is defined analogously to the intensional predicate hasAdSpace with the exception of the alternative date the ad is to be published on according to the prior agreement concerning alternative ad space between the Media Agency and Publication.

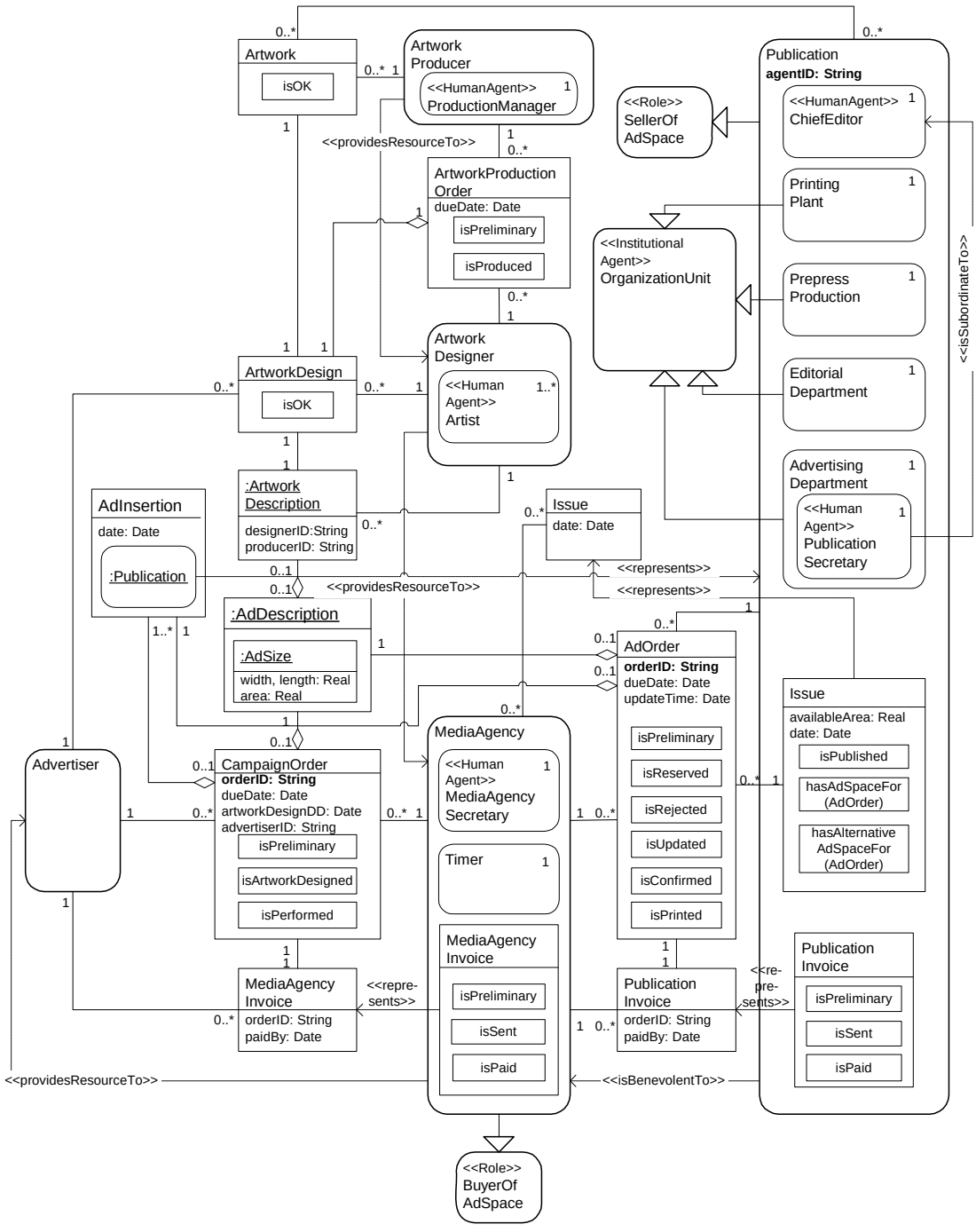


Figure 4-5. The organization and information models of the advertising case study.

4.2.4.2. Interaction Modelling

The interaction frames in the extended interaction frame diagram depicted in Figure 4-6 model the interactions described by the use cases represented in Tables 4-28 – 4-52.

The first communicative action event type in the interaction frame of the business process type of carrying out an advertising campaign models a request by the Advertiser, which can be any company, to the MediaAgency to have the advertising campaign performed. The campaign is described by the instance of CampaignOrder that is contained by the data item ?CampaignOrder of the corresponding communicative action event. Since there is a providesResourceTo relationship between the agent types MediaAgency and Advertiser in the organization model represented in Figure 4-5, the MediaAgency always agrees to have the campaign performed and forms a *see-to-it-that* commitment of the type achieve(isPerformed(?CampaignOrder) ?Date) towards the Advertiser. When the advertising campaign has been performed, the MediaAgency sends to the Advertiser a MediaAgencyInvoice which creates for the MediaAgency a claim against the Advertiser that it would pay for the campaign. The claim is satisfied through actual paying for the campaign. The invoicing process described is modelled through the corresponding action event types.

The interaction frame between the agent types MediaAgency and Publication describes the business process type of publishing an ad. In accordance with the isBenevolentTo relationship between the agent types Publication and MediaAgency, room is left for refusals and negotiations regarding ad space between agents of the types MediaAgency and Publication. The first communicative action event type of the interaction frame models a request by the MediaAgency to the Publication to reserve ad space for the ad described by the data element ?AdOrder of the action event. In reply to a message of this type, the Publication either proposes an ad space to be reserved or refuses the ad space reservation request by using the respective communicative action event type propose or refuse. If the MediaAgency receives a proposal for ad space reservation, it, in turn, either accepts or rejects it by creating a communicative action event of the type accept-proposal or reject-proposal, respectively. In case of the acceptance, the ad space reservation can be updated by exchanging messages of the types request achieve(isUpdated (?AdOrder)) and inform(isUpdated(?AdOrder)). The ad order is actually submitted by sending a message of the type request providePrintedAd(?AdOrder) from the MediaAgency to the Publication by which the MediaAgency requests a "hard copy" of the publication's issue containing the printed ad. This message is followed by sending a confirmation message of the type agree providePrintedAd(?AdOrder) in the opposite direction. At this stage, the Publication can not any more refuse the ad order. The occurrence of an action event of the type providePrintedAd(?Issue) between the Publication and MediaAgency, which models the physical delivery of the issue of the publication where the printed ad has appeared, is preceded by the formation of the corresponding commitment/claim of the type providePrintedAd(?Issue ?Date). The occurrence of a communicative action event of the type request payForAd(?PublicationInvoice) represents sending an invoice from the Publication to the MediaAgency. It results in a commitment/claim of the type payForAd(?PublicationInvoice ?Date) which is satisfied by the occurrence of an action event of the corresponding type initiated by the MediaAgency.

The first action event type between the MediaAgency and ArtworkDesigner models a request by the MediaAgency to design the artwork which is accompanied by the corresponding CampaignOrder containing the description of the artwork. In our simplified models, this request is always followed by a communicative action event of the type agree achieve(isArtworkDesigned(?CampaignOrder)) by the ArtworkDesigner, representing an agreement to have the artwork designed, and by the formation of the corresponding *stit*-commitment/claim of the type achieve(isArtworkDesigned(?CampaignOrder) ?Date) between the ArtworkDesigner and MediaAgency. The commitment is satisfied by actual completion of the artwork design. The next communicative action event type models a request from the ArtworkDesigner to the ArtworkProducer to provide the Publication, which is referred to by the identifier attribute agentID, with the artwork. This request is also never refused because of the providesResourceTo relationship between the agent types involved in the organization model in Figure 4-5. After receiving a request to provide the artwork, the ArtworkProducer commits towards the Publication to provide it with the artwork. Consequently, here the creditor of the commitment to perform an action (Publication) is different from the agent who requested it's performing (ArtworkDesigner).

In addition to the interaction frames described, there are interaction frames related to sending and approving/rejecting proofs. The ArtworkDesigner and ArtworkProducer send proofs to the Advertiser and ArtworkDesigner, respectively, and receive *yes-no* replies from them. A proof-related business process is repeated as long as the proof is accepted. The square brackets in the inform([not](isOK(?ArtworkDesign))) and inform ([not](isOK(?Artwork))) communicative action event types stand for optionality.

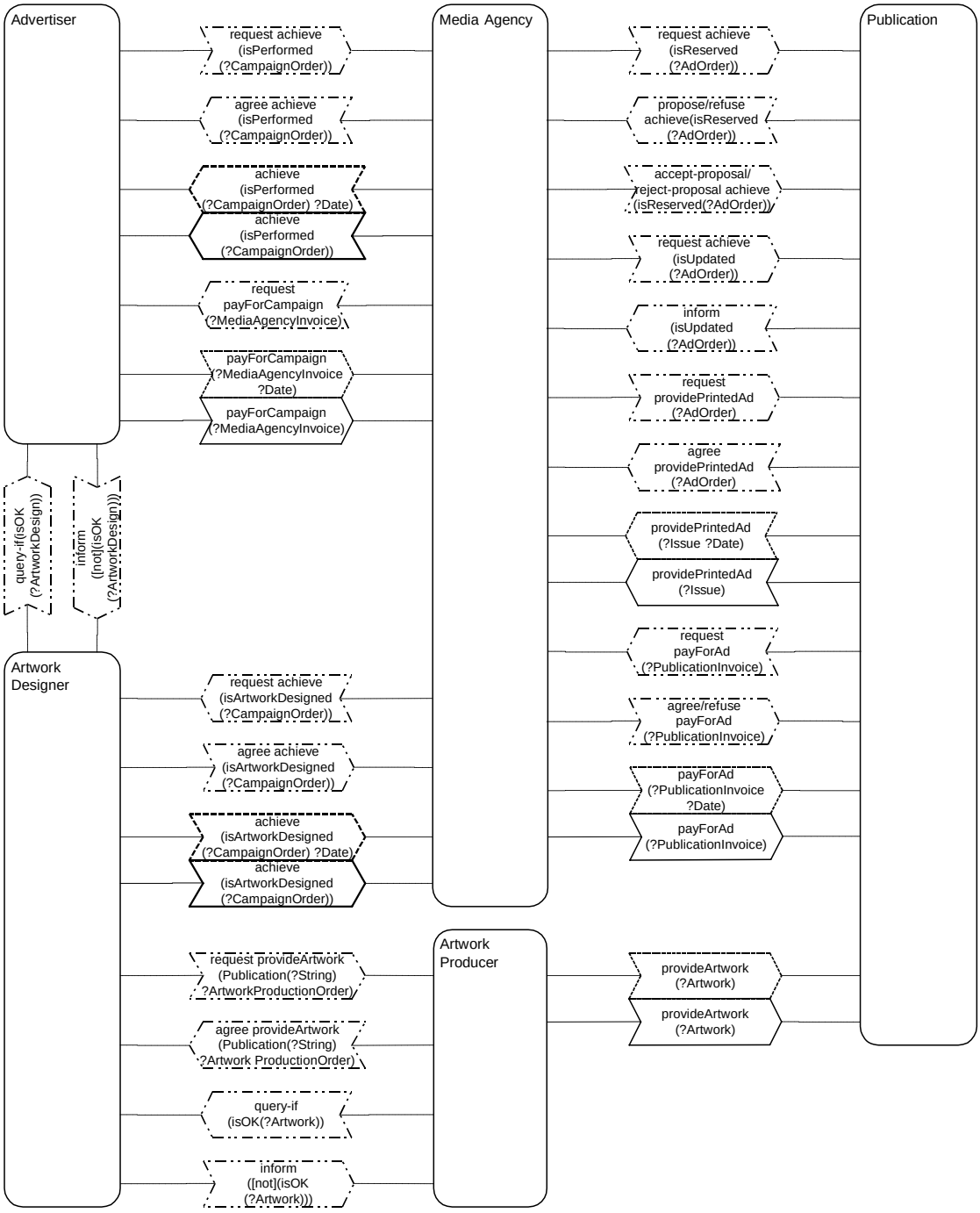


Figure 4-6. The extended interaction frame diagram of the advertising domain.

4.2.4.3. Function and Goal Modelling

By following guideline 1 of the recursive procedure described in section 3.8.4.1, the main scenarios of the descriptions of the business processes of advertising by the goal-based use cases presented in Tables 4-28 – 4-52 are turned into the corresponding activity types of the proper agent types. In Appendix G, the activity types distinguished at the stage of function and motivation modelling are presented in bold.

Firstly, the main scenarios of the primary tasks triggered by external agents are modelled according to guideline 1 in section 3.8.4.1. For example, the main scenarios of use cases 1 ("Carry out the advertising campaign") and 8 ("Complete the advertising campaign") with the media agency in focus are represented in Appendix G as the respective activity types "Manage advertising campaign" and "Complete advertising campaign" of the agent type MediaAgency. In the same way, the main scenarios of use cases 12 ("Reserve ad space") and 15 ("Have the ad printed") with the publication in focus are turned in Appendix G into the respective activity types "Manage ad space reservation" and "Manage ad publishing" of the agent type Publication. According to guideline 2 in section 3.8.4.1, the triggers of the primary tasks mentioned are modelled in Appendix G as the respective reaction rules R1, R22, R33, and R39. The main scenarios of use cases 18 ("Design the artwork") and 23 ("Produce the artwork") with the artwork designer and artwork producer in focus, respectively, are represented in Appendix G as the activity types "Manage artwork design" and "Manage artwork production" of the respective internal agent types ArtworkDesigner and ArtworkProducer. The triggers of these primary tasks are modelled in Appendix G as the respective reaction rules R51 and R67.

Next, subordinate use cases (subfunctions) of the primary tasks are turned into the respective sequential subactivity types as is described in guidelines 3 – 5 in section 3.8.4.1. The subfunctions "Request artwork design" (Use Case 2), "Have ad space reserved in the publications" (Use Case 3), "Have the ad space reservation updated" (Use Case 6), and "Request printing of the ad" (Use Case 7) of the primary task "Carry out the advertising campaign" (Use Case 1) are thus modelled as the respective sequential subactivity types "Manage artwork design", "Manage ad space reservations", "Have the ad space reservation updated", and "Request printing of the ad" of the activity type "Manage advertising campaign". All the activity types mentioned belong to the agent type MediaAgency. The subactivity types "Have the ad space reservation updated" and "Request printing of the ad" are triggered by the respective internal agents :MediaAgencySecretary and :Timer.

Following the same guidelines, for the activity type "Manage ad space reservation" of the agent type Publication, corresponding to the primary task "Reserve ad space" (Use Case 12), is distinguished the subactivity type "Wait for and process reply", corresponding to the primary task's subfunction "Process the reply by the media agency" (Use Case 13). The subfunctions "Receive the artwork" (Use Case 16) and "Deal with the publication invoice" (Use Case 17) of the primary task "Have the ad printed" (Use Case 15) are modelled as the respective sequential subactivity types "Receive and insert artwork" and "Manage ad invoicing" of the agent type Publication.

The process described by guidelines 3 – 5 in section 3.8.4.1 is recursively repeated for all subfunctions and steps of a primary task as long as the modelling precision of the desired level is achieved. If the step of a scenario does not include any subfunctions, it is modelled as an elementary activity type. For example, the four steps of the main scenario of the subfunction "Deal with the publication invoice" (Use Case 17) are modelled as the elementary activity types "Provide printed ad", "Update ad size", "Create publication invoice", and "Send publication invoice" of the agent type Publication.

According to guidelines 3 – 5 in section 3.8.4.1, the activity type "Manage artwork design" of the agent type ArtworkDesigner, corresponding to the primary task "Design the artwork", is modelled as the sequence of two activity types: the elementary activity type "Confirm and commit to design artwork" and the subactivity type "Register artwork design". Analogously, the activity type "Manage artwork production" is modelled as consisting of the elementary activity type "Confirm and commit to produce artwork" and the subactivity type "Register artwork production" next to it.

In function and motivation models we do not represent activities that are repeated for each instance of some (possibly constrained by a predicate) informational entity type because activity diagrams of function and motivation models do not lend themselves to expressing conditions (of repeating). For example, as Appendix G shows, the activity types "Manage ad order" of MediaAgency and "Manage publication" of ArtworkDesigner, whose instances are repeated for each instance of AdOrder and Publication, respectively, are not represented in function and motivation models.

As we explained in section 3.8.4.2, preconditions and goals are defined for activity types as propositions by means of OCL. Table 4-53 includes the precondition and goal defined for each activity type represented by the function model of the case study of advertising.

The precondition defined for the activity type "Manage advertising campaign" of the agent type MediaAgency in Table 4-53 means that within an agent of this type exists the instance of CampaignOrder that (1) has the status isPreliminary, (2) has the value of its attribute agentID equal to the value of the activity's input parameter senderID, and (3) is referred to by the value of the activity's input parameter co. The meaning of the goal defined for the same activity type is that all the advertising orders included by the CampaignOrder are either confirmed or rejected and the MediaAgency has the *see-to-it-that* commitment towards the Advertiser to perform the campaign. The latter is reflected by achieving the status isPerformed of the corresponding instance of CampaignOrder. According to the derivation rule defined in section 4.2.4.1, an instance of CampaignOrder has the status isPerformed if each instance of AdOrder connected to it through the instance of AdInsertion shared by both has the status isRejected or isPrinted. The status predicate isPrinted of an instance of AdOrder is true if the publishing of the ad corresponding to the AdOrder has been registered.

Analogously, the precondition defined for the activity type "Manage ad space reservation" of the agent type Publication is the existence of the corresponding instance of AdOrder that has the status isPreliminary and is referred to by the value of the activity's input parameter co. Additionally, the object mentioned has to be associated with the instance of the Publication's private object type Issue, having the same date as the AdInsertion included by the AdOrder, and with the instance of the representation of the agent type MediaAgency, identified by the value of the input parameter senderID. The goal defined for the activity type "Manage ad space reservation", which is represented in Table 4-53, is that the instance of AdOrder has either the status isReserved or the status isRejected.

The precondition defined for the activity type "Manage ad publishing" of the agent type Publication is that within an agent of the type Publication exists the instance of AdOrder with the status isReserved or isUpdated that is referred to by the value of the activity's input parameter ao of the type AdOrder. The goal defined for the same activity type specifies that (1) the ad specified by the advertising order has been printed, which is reflected by the status isPrinted of the corresponding instance of AdOrder, (2) the PublicationInvoice has been sent to the media agency, which is reflected by its status isSent, and (3) the corresponding *see-to-it-that* claim against the MediaAgency to pay for the ad according to the invoice has been created, and optionally, if the ad order includes the description of the artwork, (4) the artwork associated with the corresponding artwork description exists.

The definition of the goal of an agent of the type ArtworkDesigner to design the artwork is attached to the corresponding activity type "Manage artwork design". The goal is defined as the conjunction of the expressions for the desired status isArtworkDesigned of the corresponding instance of CampaignOrder and the existence of the instance of ArtworkProductionOrder associated with the corresponding ArtworkDesign. According to the derivation rule defined in section 4.2.4.1, an instance of CampaignOrder has the status isArtworkDesigned if there is the instance of ArtworkDesign connected to it with the status isOK, which is the case after the proof of the artwork design has been accepted by the advertiser. The precondition defined for the activity type "Manage artwork design" is that within the ArtworkDesigner exists the instance of CampaignOrder with the status isPreliminary that is associated with the corresponding instance of the representation of the agent type MediaAgency, and is referred to by the value of the activity's input parameter co.

Analogously, the precondition defined for the activity type "Manage artwork production" specifies the existence of the instance of ArtworkProductionOrder that is associated with the instance of the representation of the agent type ArtworkDesigner. The latter is identified by the value of the input parameter senderID. The goal defined for the activity type "Manage artwork production" is that the artwork specified by the artwork production order has been produced which is reflected by the status predicate isProduced of the ArtworkProductionOrder. The latter is defined in section 4.2.4.1.

Table 4-53. Activities of the case study of advertising with their preconditions and goals.

Activity type and input parameter(s)	Precondition	Goal
Manage advertising campaign (co : CampaignOrder, senderID: String)	CampaignOrder. allInstances->exists (c : CampaignOrder | c.isPreliminary and c.advertiserID = senderID and co = c)	co.adInsertion->forAll (adOrder->exists (ao : AdOrder | ao = co.adInsertion and (ao.isConfirmed or ao.isRejected))) **and** self.stitCommitmentClaim->exists (achieve = co.isPerformed and dueTime = co.dueDate and sourceID = self.agentID and targetID = senderID)
Agree and commit to perform campaign	-	self.stitCommitmentClaim->exists (achieve = co.isPerformed and dueTime = co.dueDate and sourceID = self.agentID and targetID = senderID)
Manage artwork design	-	-
Send artwork design request	-	-
Receive agreement	-	-
Manage ad space reservations	-	co.adInsertion->forAll (adOrder->exists (ao : AdOrder | ao = co.adInsertion and (ao.isConfirmed or ao.isRejected)))
Have the ad space reservation updated (ao : AdOrder)	AdOrder.allInstances->exists (o : AdOrder | o.isReserved or o.isUpdated and ao = o)	ao.isUpdated
Request update	-	-
Register update	-	ao.isUpdated
Request printing of the ad (ao : AdOrder)	AdOrder.allInstances->exists (o : AdOrder | o.isReserved or o.isUpdated and ao = o)	ao.isConfirmed
Request printing	-	-
Register confirmation	-	ao.isConfirmed
Complete advertising campaign (ao : AdOrder, sourceID : String)	AdOrder.allInstances->exists (o : AdOrder | o.isConfirmed and ao = o)	ao.isPrinted **or** (ao.isPrinted **and** ao.adDescription.campaignOrder. isPerformed **and** MediaAgencyInvoice.allInstances->exists (mi: MediaAgencyInvoice | mi.isSent and mi.orderID = co.orderID and co.mediaAgencyInvoice = mi and mi.campaignOrder = co) **and** payForCampaign.allInstances->exists (about = ao.adDescription.campaignOrder. mediaAgencyInvoice and dueTime = ao.adDescription.campaignOrder. mediaAgencyInvoice.paidBy and sourceID = ao.adDescription.campaignOrder.advertiser. agentID and targetID = self.agentID))
Register printing of the ad	-	ao.isPrinted **or** (ao.isPrinted **and** ao.adDescription.campaignOrder.isPerformed)

Table 4-53 (continued). Activities of the case study of advertising with their preconditions and goals.

Manage media agency invoice (co : CampaignOrder)	CampaignOrder. allInstances->exists (c : CampaignOrder | c.isPerformed and c = ao.adDescription.campaignOrder and co = c)	MediaAgencyInvoice.allInstances->exists (mi: MediaAgencyInvoice | mi.isSent and mi.orderID = co.orderID and co.mediaAgencyInvoice = mi and mi.campaignOrder = co) **and** payForCampaign.allInstances->exists (about = co.mediaAgencyInvoice and dueTime = co.mediaAgencyInvoice.paidBy and sourceID = co.advertiser.agentID and targetID = self.agentID)
Create media agency invoice	-	MediaAgencyInvoice.allInstances->exists (mi: MediaAgencyInvoice | mi.isPreliminary and mi.orderID = co.orderID and co.mediaAgencyInvoice = mi and mi.campaignOrder = co)
Send media agency invoice	-	co.mediaAgencyInvoice.isSent **and** payForCampaign.allInstances->exists (about = co.mediaAgencyInvoice and dueTime = co.mediaAgencyInvoice.paidBy and sourceID = co.advertiser.agentID and targetID = self.agentID)
Register the payment (invoice : MediaAgencyInvoice, originID : String)	MediaAgencyInvoice-exists (i : MediaAgencyInvoice | i.isSent and invoice = i)	invoice.isPaid **and** **not**(payForCampaign.allInstances->exists (about = invoice and dueTime = invoice.paidBy and sourceID = originID and targetID = self.agentID))
Deal with publication invoice (pi: PublicationInvoice)	PublicationInvoice. allInstances->exists (i : PublicationInvoice | i.adOrder->exists (o: AdOrder | o.orderID = i.orderID and o.publicationInvoice = i) and pi = i)	payForAd.allInstances->exists (about = pi and dueTime = pi.paidBy and sourceID = self.agentID and targetID = senderID)
Manage ad space reservation (ao: AdOrder, senderID : String)	AdOrder.allInstances->exists (o : AdOrder | o.isPreliminary and o.issue->exists (is: Issue | is.date = o.adInsertion.date and is.adOrder->includes(o)) and o.mediaAgency->exists (ma: MediaAgency | ma.agentID = senderID and ma.adOrder->includes(o)) and ao = o)	ao.isReserved **or** ao.isRejected
Wait for and process reply	-	ao.isReserved **or** ao.isRejected
Update ad space reservation (ao : AdOrder, senderID : String)	AdOrder.allInstances->exists (o : AdOrder | o.isReserved or o.isUpdated and ao = o)	ao.isUpdated

Table 4-53 (continued). Activities of the case study of advertising with their preconditions and goals.

Manage ad publishing (ao : AdOrder, senderID : String)	AdOrder.allInstances->exists (o : AdOrder | o.isReserved or o.isUpdated and ao = o)	(ao.isPrinted **and** PublicationInvoice.allInstances->exists (i : PublicationInvoice | i.isSent **and** i.adOrder = ao **and** ao.publicationInvoice = i) **and** payForAd.allInstances->exists (about = ao.publicationInvoice and dueTime = ao.publicationInvoice.paidBy and sourceID = ao.mediaAgency.agentID **and** targetID = self.agentID)) **or** (ao.isPrinted **and** PublicationInvoice.allInstances->exists (i : PublicationInvoice | i.isSent **and** i.adOrder = ao **and** ao.publicationInvoice = i) **and** payForAd.allInstances->exists (about = ao.publicationInvoice and dueTime = ao.publicationInvoice.paidBy and sourceID = ao.mediaAgency.agentID and targetID = self.agentID) **and** Artwork.allInstances->exists (aw: Artwork | aw.artworkDescription = ao.adDescription.artworkDescription and ao.adDescription.artworkDescription = aw))
Confirm ad order	-	ao.isConfirmed **and** providePrintedAd.allInstances->exists (about = ao.issue and dueTime = ao.dueDate and sourceID = self.agentID and targetID = senderID)
Receive and insert artwork (awd: ArtworkDescription)	ArtworkDescription. allInstances->exists (d: ArtworkDescription | d = ao.adDescription. artworkDescription and awd = d)	Artwork.allInstances->exists (aw: Artwork | aw.artworkDescription = awd and awd.artwork = aw)
Manage ad invocing (ao : AdOrder)	AdOrder.allInstances->exists (o : AdOrder | o.isConfirmed and ao = o)	ao.isPrinted **and** **not**(providePrintedAd.allInstances->exists (about = ao.issue and dueTime = ao.dueDate and sourceID = self.agentID and targetID = ao.mediaAgency.agentID)) **and** PublicationInvoice.allInstances->exists (i : PublicationInvoice | i.isSent and i.adOrder = ao and ao.publicationInvoice = i) **and** payForAd.allInstances->exists (about = ao.publicationInvoice and dueTime = ao.publicationInvoice.paidBy and sourceID = ao.mediaAgency.agentID and targetID = self.agentID)
Provide printed ad	-	ao.isPrinted **and** **not**(providePrintedAd.allInstances->exists (about = ao.issue and dueTime = ao.dueDate and sourceID = self.agentID and targetID = ao.mediaAgency.agentID))
Update ad size	-	-
Create publication invoice	-	PublicationInvoice.allInstances->exists (i : PublicationInvoice | i.isPreliminary and i.adOrder = ao and ao.publicationInvoice = i)

Table 4-53 (continued). Activities of the case study of advertising with their preconditions and goals.

Send publication invoice (pi : PublicationInvoice)	PublicationInvoice. allInstances->exists (i : PublicationInvoice | i.isPreliminary and i = ao.publicationInvoice and pi = i)	pi.isSent **and** payForAd.allInstances->exists (about = pi and dueTime = pi.paidBy and sourceID = ao.mediaAgency.agentID and targetID = self.agentID)
Manage artwork design (co : CampaignOrder, senderID : String)	CampaignOrder. allInstances->exists (o : CampaignOrder | o.isPreliminary and o.mediaAgency->exists (ma: MediaAgency | ma.agentID = senderID and ma.campaignOrder->includes(o)) and co = o)	co.isArtworkDesigned **and** Artwork.allInstances->exists (a : Artwork | a.artworkDesign = co.adDescription.artworkDescription. artworkDesign and co.adDescription.artworkDescription. artworkDesign.artwork = a) **and** ArtworkProductionOrder.allInstances->exists (apo: ArtworkProductionOrder | apo.isProduced and apo.artworkDesign = co.adDescription.artworkDescription. artworkDesign and co.adDescription.artworkDescription. artworkDesign.artworkProductionOrder = apo)
Confirm and commit to design artwork	-	self.stitCommitmentClaim->exists (achieve = co.isArtworkDesigned and dueTime = co.artworkDesignDD and sourceID = self.agentID and targetID = senderID)
Register artwork design (ad : ArtworkDesign, co : CampaignOrder)	CampaignOrder. allInstances->exists (o : CampaignOrder | o.isPreliminary and co = o)	co.isArtworkDesigned **and** co.adDescription. artworkDescription.artworkDesign = ad **and not** (self.stitCommitmentClaim->exists (achieve = co.isArtworkDesigned and dueTime = co.artworkDesignDD and sourceID = self.agentID and targetID = co.mediaAgency.agentID)) **and** ArtworkProductionOrder.allInstances->exists (apo: ArtworkProductionOrder | apo.isProduced and apo.artworkDesign = ad and ad.artworkProductionOrder = apo) **and** Artwork.allInstances->exists (aw : Artwork | aw.artworkDesign = ad and ad.artwork = aw)
Store artwork design	-	ArtworkDesign.allInstances->exists (d : ArtworkDesign | d.artworkDescription = co.adDescription. artworkDescription and co.adDescription. artworkDescription.artworkDesign = d and ad = d)
Have artwork produced	-	ArtworkProductionOrder.allInstances->exists (apo: ArtworkProductionOrder | apo.isProduced and apo.artworkDesign = ad and ad.artworkProductionOrder = apo) **and** Artwork.allInstances->exists(a : Artwork | a.artworkDesign = ad and ad.artwork = a)
Create artwork production order	-	ArtworkProductionOrder.allInstances->exists (apo: ArtworkProductionOrder | apo.isPreliminary and apo.artworkDesign = ad and ad.artworkProductionOrder = apo)

Table 4-53 (continued). Activities of the case study of advertising with their preconditions and goals.

Request production	-	-
Receive agreement		-
Receive proof	-	Artwork.allInstances->exists(a : Artwork | a.artworkDesign = ad and ad.artwork = a)
Manage artwork distribution	-	-
Manage artwork production (po : Artwork ProductionOrder, senderID : String)	ArtworkProductionOrder. allInstances->exists (o : ArtworkProductionOrder | o.isPreliminary and o.artworkDesigner->exists (ad: ArtworkDesigner | ad.agentID = senderID and ad.artworkProductionOrder-> includes(o)) and po = o)	po.isProduced
Confirm and commit to produce artwork	-	self.stitCommitmentClaim->exists (achieve = po.isProduced and dueTime = po.dueDate and sourceID = self.agentID and targetID = senderID)
Register artwork production (aw : Artwork, po : Artwork ProductionOrder)	ArtworkProductionOrder. allInstances->exists (p : ArtworkProductionOrder | p.isPreliminary and po = p)	po.isProduced **and** po.artworkDesign.artwork = aw **and not** (self.stitCommitmentClaim->exists (achieve = po.isProduced and dueTime = po.dueDate and sourceID = self.agentID and targetID = po. artworkDesigner.agentID))
Store artwork	-	Artwork.allInstances->exists (a : Artwork | a.artworkDesign = po.artworkDesign and po. artworkDesign.artwork = a and aw = a)
Provide the publication with the artwork (publicationID : String, po : Artwork ProductionOrder, senderID : String)	ArtworkProductionOrder. allInstances->exists (o : ArtworkProductionOrder | o.isProduced and po = o)	-
Agree and commit to provide artwork	-	provideArtwork.allInstances->exists (about = po.artworkDesign.artwork and dueDate = now() + 1 and sourceID = self.agentID and targetID = publicationID)[14]
Provide artwork	-	**not**(provideArtwork.allInstances->exists (about = po.artworkDesign.artwork and dueDate = now() + 1 and sourceID = self.agentID and target ID = publicationID))

[14] Since the *to-do*-commitment created here is discharged almost right away, if everything goes normally, to its attribute dueDate is assigned the date of the next day.

4.2.4.4. Behaviour Modelling

At the step of behaviour modelling, function and motivation models of business processes by the goal-based use cases presented in Tables 4-28 – 4-52 are transformed into behaviour models by following the guidelines provided in section 3.8.5.2.

According to guideline 1, the <*time or sequence factor*> component "*The advertising campaign is authorized by the media agency secretary*" of step 1 of use case 1 ("Carry out the advertising campaign") is modelled in Appendix G as the respective communicative action event type authorizeCampaign connected to reaction rule R2. The latter is triggered by the internal agent :MediaAgencySecretary. Following the same guideline, the <*time or sequence factor*> components "*A request to update the ad space reservation by the media agency secretary is received* " and "*A request by the timer to request printing of the ad is received*" of steps 4 and 5 of use case 1 are represented as the non-communicative action event types updateAdOrder(?AdOrder) and request achieve (isPrinted(AdOrder(?String))) connected to reaction rules R16 and R19, respectively. The data element ?String in the latter action event type contains the value of the identifier attribute orderID of the corresponding instance of AdOrder. In the activity diagrams presented in Appendix G, the relevant instances of AdOrder are identified by the precondition arrows which lead to the symbols of both last mentioned reaction rules. The precondition arrow along with the OCL expression possibly attached to it define the precise scope of a reaction rule, i.e. the set of entity instances that the action and postcondition parts of the rule affect according to the principles laid out in section 3.6.4.

Behaviour modelling based on guideline 1 also enables to represent the behavioural pattern "Deferred choice" which is described in section 3.8.5.3. For example, the alternative steps 1 and 1a of use case 13 ("Process the reply by the media agency") and 5 and 5a of use case 20 ("Have the artwork produced") are modelled in Appendix G as the respective pairs of reaction rules R35 and R36, and R62 and R63 which are triggered by an external agent of the type MediaAgency and an internal agent of the type Artist, respectively.

According to guideline 2 presented in section 3.8.5.2, the <*condition*> component "*The campaign order includes an artwork description*" of step 2 of use case 1 ("Carry out the advertising campaign") is modelled in Appendix G as reaction rule R4 that invokes an activity of the type "Manage artwork design" if the CampaignOrder includes the ArtworkDescription or an activity of the type "Manage ad space reservations" in the opposite case. Analogously, the <*condition*> component "*There is sufficiently ad space or alternative ad space for the reservation request in question*" of step 1 of use case 12 ("Reserve ad space") is represented in Appendix G by reaction rule R34 that invokes an activity of the type "Wait for and process reply" and sends to the MediaAgency the corresponding message of the type propose achieve(isReserved(?AdOrder)) if there is enough ad space or alternative ad space for the ad described by the AdOrder. In the opposite case, according to the same reaction rule a refusal message is sent to the MediaAgency and the refusal is registered within the Publication. The symbol for reaction rule R4 as well as that for reaction rule R34 includes the precondition arrow augmented by an OCL expression that defines the precise scope of the precondition, as is described in section 3.6.4.

Based on the same guideline, the <*condition*> components included by use cases 3 ("Have ad space reserved in the publications") and 21 ("Have the artwork distributed") are turned in Appendix G into reaction rules R7 and R64, respectively, which form the corresponding "For-Each" loop patterns described in section 3.8.5.3.

According to guideline 4 provided in section 3.8.5.2, a precondition arrow of either reaction rule mentioned above is augmented by an OCL equation limiting the set of instances of the informational entity type for which the "For-Each" loop is performed. The OCL expression {campaignOrder = co} attached to the precondition arrow of reaction rule R7 specifies that the action part of the rule (starting an activity of the type "Manage ad order") is repeated for each instance of AdInsertion that is included by the instance of CampaignOrder that is referred to by the value of the input parameter co. In the same way, the OCL expression {adInsertion.campaignOrder = co} pertaining to the precondition of reaction rule R64 specifies that the action part of the rule (starting an activity of the type "Manage publication") is repeated for each instance of Publication that is included by the AdInsertion forming a part of the instance of CampaignOrder that is referenced by the input parameter co.

Following guideline 3, in Appendix G the symbol for the communicative action event type agree achieve (isPerformed(?CampaignOrder)) is connected to reaction rule R3 included by the elementary

activity type “Agree and commit to perform campaign”. In the same way, the symbols for the communicative action event types request achieve (isUpdated(?AdOrder))) and inform(isUpdated(?AdOrder)) are connected to the respective reaction rules R17 and R18 included by the elementary activity types “Request update” and “Register update”, respectively. The mental effect arrows originating in the symbols for the activity types “Agree and commit to perform campaign” and “Register update” are also connected to reaction rules R3 and R18, respectively. Analogously, the symbol for the non-communicative action event type updateAdSize and the arrow standing for the accompanying mental effect are connected to reaction rule R45 that is included by the elementary activity type “Update ad size”.

As we explained in section 3.6.5, a mental effect arrow of a reaction rule may be augmented by an OCL expression that (re)defines the mental effect of the rule. For example, the OCL expression {advertiserID = SenderID} attached to the mental effect arrow of reaction rule R1 in Appendix G determines that the value of the attribute advertiserID of the instance of CampaignOrder to be created by the rule should be equal to the value of the reaction rule’s internal variable SenderID.

According to guideline 4 provided in section 3.8.5.2, also a precondition arrow may be augmented by an OCL expression. For example, in addition to specifying the creation of an instance of AdOrder, the augmented precondition arrow and the mental effect arrows of reaction rule R30 in Appendix G determine the association to be created between the instance of PublicationInvoice within the scope of the rule and the instance of AdOrder, identified by the value of its attribute orderID which must be equal to the value of the attribute of the same name of the PublicationInvoice. Analogously, the precondiiton and the mental effects of reaction rule R42 determine that there should be a two-way association between the instance of Artwork to be created and the :ArtworkDescription that is referred to by the value of the input parameter awd. As another example, the precondition and mental effects of reaction rule R58 determine that (1) the instance of ArtworkProductionOrder created has the status isPreliminary, (2) the instance of ArtworkProductionOrder created includes the instance of ArtworkDesign that is referred to by the value of the input parameter ad, and (3) the ArtworkDesign mentioned also refers to the instance of ArtworkProductionOrder created.

As was explained in section 3.8.5.3, an elementary activity type is not always needed because the required actions and mental effects may be invoked and achieved, respectively, by a reaction rule which is not included by any enclosing elementary activity type. This feature is capable of making activity diagrams smaller and more compact. For example, reaction rule R34 in Appendix G sends by itself the messages to be sent and achieves the mental effects to be achieved.

5. CONCLUSIONS AND OUTLOOK

5.1. THESIS SUMMARY

In section 1.6 we have stated that the goal of this thesis is to work out and apply a modelling notation and methodology that enable to create and integrate business models of different perspectives. The modelling technique should be usable at the analysis and design stages of business modelling and it should lend itself to the creation of executable business process models. Another objective of the dissertation has been declared in section 1.6, which is laying a cornerstone for a systematic development approach of AOIS and CIS.

In order to meet these goals, we have worked out a modelling notation and methodology that support business modelling under all six views of agent-oriented modelling proposed by us in section 1.5.6. Moreover, the modelling notation developed by us enables to represent the integrated models of all six views in just one diagram.

Since, as we showed in section 1.5.7, reaction rules span all six views of agent-oriented modelling, we have based our modelling notation on AORML which includes modelling by reaction rules. To enable adequate modelling of business processes, we have extended AORML by activity diagrams where activities are started and controlled by reaction rules. Executability of activity diagrams has been ensured by extending the semantic framework of KPMC agents, that the operational semantics for reaction rules is based on, with the operational semantics for activities. For the modelling of derivation rules, and preconditions and goals of activities, we have adapted and, when necessary, extended OCL.

We have also worked out a methodology which consists of the steps of analysis by goal-based use cases and design by using the extended AORML. The design step, in turn, consists of the stages of organization modelling, information modelling, interaction modelling, function and motivation modelling, and behaviour modelling which correspond to the six views of agent-oriented modelling.

The modelling methodology proposed by us also serves as a systematic approach to the development of AOIS and CIS because it is based on the six views of agent oriented modelling. As empirical proofs of this serve the case studies of the ceramic factory and advertising which are aimed at creating distributed agent-oriented information systems.

Finally, we brought reaction rules and activities straightforwardly to the implementation level and demonstrated how they can be mapped to the notions of the JADE agent platform and simulated there

One result achieved in the thesis came as a surprise to even ourselves. Namely, we did not expect a reaction rule proving to be so powerful construct for defining an agent's behaviour. One may ask: are behavioural rules true business rules any more? Our answer is: they are because they determine an agent's behaviour *based on its knowledge state* represented in its VKB which serves as an abstraction of the internal information systems of a company containing business data.

In summary, the business modelling methodology of the Business Agents' Approach proposed by us in this thesis consists of the following steps:

1. **Analysis**:
   - 1.1. Sketch the business agent types and instances (if applicable) of the problem domain.
   - 1.2. Model the activities of the business agents by goal-based use cases.
2. **Design**:
   - 2.1. Create the organization model of the problem domain by using an agent diagram of the extended AORML.
   - 2.2. Create the information model of the problem domain by using a combination of an agent diagram of the extended AORML and extended OCL.
   - 2.3. Create the interaction models of the problem domain by using interaction frame diagrams of the extended AORML.
   - 2.4. Create the function and motivation models of the problem domain by using activity diagrams of the extended AORML and extended OCL expressions.
   - 2.5. Refine the function models into the corresponding behaviour models by using activity diagrams of the extended AORML and extended OCL expressions.

5.2. COMPARISON TO OTHER APPROACHES

In sections 2.1 through 2.3, we conducted a comparative study of eight business modelling techniques that are related to agents/actors and/or business rules. Based on the evaluation of these techniques, the need for a distinctive technique of agent-oriented modelling was identified. In Table 5-1, we compare such a technique devised by us – the Business Agents' Approach – with the business modelling techniques that were reviewed and evaluated in Chapter 2.

In addition to incorporating all the best features from UML, information modelling in the Business Agents' Approach complements UML by proposing extensions to OCL that enable to represent properly all kinds of derivation rules – derived attributes, status predicates, and intensional predicates. Moreover, while e.g. the Eriksson-Penker Business Extensions to UML do not present a general method how to express derivation rules in a class diagram, we demonstrated in section 3.8.2.1 how derivation rules can be represented and visualized. The informational view of agent-oriented modelling is thus very strongly supported in the Business Agents' Approach.

Organization modelling in the Business Agents' Approach explicitly supports representing institutional agents and their internal human and/or artificial agents. The organization modelling also distinguishes between institutional and human roles. In addition, the organization modelling comprises modelling of inheritance, aggregation, subordination, control, benevolence, and dependency relationships between agent types and instances. Consequently, organization modelling in our approach is more precise than e.g. in the Eriksson-Penker Business Extensions and CIMOSA.

Interaction modelling in the Business Agents' Approach is based on the control, benevolence, and dependency relationships. In addition to enabling precise communication modelling based on speech acts, the Business Agents' Approach also provides the means for modelling other forms of interaction between agents, like delivering a physical object. Moreover, our approach also includes the notions of deontic modelling such as commitments and claims and operations upon them. The support for the interactional view can thus also be regarded as very strong in our approach.

Function modelling in the Business Agents' Approach enables to represent function hierarchies and control and data flows by means of activity diagrams. The motivational view is supported by assigning preconditions and goals defined in OCL to activities performed by business agents. Since the Business Agents' Approach for the time being lacks the reasoning mechanism for justifying goals, and goals are not assigned to activities at runtime, the motivational view is supported a bit more weakly than the other views.

As we saw in section 3.8.5.3, the Business Agents' Approach provides a stronger support for behavioural patterns than e.g. UML. Additionally, since the operational semantics of activity diagrams of the extended AORML is based on the extended semantic framework of KPMC agents, behaviour modelling in our approach is free from the hierarchy constraints, which were briefly described in section 2.1.2.2, that both activity diagrams of UML and behaviour models of CIMOSA suffer from. Consequently, the Business Agents' Approach provides a better support for the behavioural view than the Eriksson-Penker Business Extensions and CIMOSA, let alone the other modelling techniques compared.

Table 5-1. Comparison of the Business Agents' Approach with the business modelling techniques studied in sections 2.1 through 2.3.

	Informational	Organizational	Interactional	Functional	Motivational	Behavioural
Ross Notation	+++	+	-	+	++	-
Eriksson-Penker Business Extensions	++	++	+	++	++	++
Role Activity Diagrams	-	++	++	+++	+	++
*i**	+	++	+	++	+++	-
CIMOSA	++	++	+	+++	+	+++
BROCOM	+++	++	-	++	-	+
EKD	++	++	+	++	++	+
GAIA	+	+	++	+++	+	+
Business Agents' Approach	+++	+++	+++	+++	++	+++

5.3. SUMMARY OF CONTRIBUTIONS

The main contributions of this thesis in the field of conceptual business modelling and design of information systems can be grouped in the following way:

- *Further systemization of the field of agent-oriented information systems:*
  - We have provided new, more precise definitions of business rules and business processes and a classification of business rules which all comply with the principles of agent-orientation (sections 1.4.1 and 1.4.2).
  - Based on the comparison and evaluation of the existing business modelling frameworks, we have proposed an improvement on them – six views of agent-oriented modelling (section 1.5.6).
  - A metamodel reflecting the six views of agent-oriented modelling has been put forward (section 3.3).
- *Promotion of agents as useful modelling abstractions that can be used at different logical levels in the modelling of organizations and their information systems, instead of mere technological building blocks*. This contribution of ours has been acknowledged e.g. in [Dignum02]. In particular:
  - Business rules have traditionally been modelled and implemented in the narrow context of (active) databases. We have adopted a broader view, and a more cognitive stance, by proposing to model and implement business rules as the "rules of behaviour" of business agents (section 3.2).
- *Extension of AORML by activity diagrams* (sections 3.6.1 – 3.6.7):
  - An operational semantics for activity diagrams has been presented by extending the semantic framework of KPMC agents with the operational semantics for activities (section 3.6.6). This ensures the executability of both function and behaviour models.
  - To the best of our knowledge, our modelling approach is the first one where partially specified function models by activity diagrams can be executed (section 3.6.6). This facilitates iterative business modelling which is state-of-the-practice.
- *Creation of a modelling process and methodology, based on the extended AORML, consisting of the steps of analysis and design* (sections 3.7 and 3.8):
  - We have suggested applying goal-based use cases to the modelling step of analysis. This is the first attempt to adapt goal-based use cases to agent-oriented modelling (section 3.7).
  - A sequence of steps for transforming goal-based use cases into activity diagrams has been presented (sections 3.8.4 – 3.8.5).
  - An original way of using the extended AORML and OCL in a combined manner as a basis for the design phase has been proposed (section 3.8).
  - To enable the modelling of derivation rules and shared object types, we have extended OCL by some additional constructs (section 3.8.2.1).
  - An interaction ontology for agents has been suggested enabling them to store and share knowledge about types and instances of messages and non-communicative action events to be created and perceived, as well as of commitments/claims in force (section 3.8.3.3).
  - We have shown that an activity diagram of the behavioural view enables to represent the models of all six views of agent-oriented modelling in just one diagram.
  - We have demonstrated that combinations of reaction rules and types of activities started and sequenced by them allow them to represent 16 out of 19 behavioural workflow patterns. The extended AORML thus provides a better support for workflow patterns than any other business modelling language or notation we are aware of (section 3.8.5.3).
- *Showing how activity diagrams can be simulated on the JADE agent platform*:
  - We have presented mappings from the notions of the extended AORML to the object classes and methods of JADE (section 3.8.6).
- *Applying the modelling methodology to the case studies of the ceramic factory and advertising*:
  - The purposes of the case study of the ceramic factory are simulating the business/manufacturing processes of the factory and preparing for the creation of a semiautomatic (agent-based) control system for the factory (section 4.1).
  - The case study of advertising serves as the first step towards (agent-based) automation of inter-enterprise business processes related to advertising (section 4.2)

5.4. LIMITATIONS AND OPEN ISSUES

Even though the extended AORML includes modelling of goals which are defined for activity types, their usage is presently limited. In particular, they are used only at the time of modelling but are not maintained and utilized by agents at runtime. Also, the Business Agents' Approach currently lacks the reasoning mechanism for justifying goals attached to activities.

As was mentioned in section 3.8.5.1, we do no treat goal-based generation of plans at runtime in this thesis. The principles how this could be done on the basis of the approach by [Fikes71] are presented in [Wagner00b]. However, the computational efficiency of the planning algorithm proposed in [Wagner00b] needs to be improved.

The feature of querying an agent's activity state, which is made possible by the operational semantics of activity diagrams, presented in section 3.6.6, is currently not used. Since, as we stated in section 3.6.1, elementary activities can be viewed as transactions, explicit transaction handling would be one way to make the most of the feature mentioned.

At present, when an activity diagram is executed, only one outermost activity can be under execution at any moment of time. We have to study how this restriction could be lifted which would allow for many parallel first-level activities.

Currently there are no integrity rules for the case when multiple activities performed by an agent need to update the same entity of the agent's VKB. One way to achieve a solution to this problem would be adopting the *isolation rule* as is suggested in [Eshuis02a]: an entity cannot be updated and either read or updated at the same time.

The treatment of commitments/claims should be elaborated on. In particular, it should be precisely specified how the proposition formula of a *stit*-commitment/claim type defines the type.

5.5. ONGOING RESEARCH WORK

In March 2003, a project called Plug-and-Trade B2B started at VTT (Technical Research Centre of Finland) Information Technology. The project lasts until the end of April 2004. The Plug-and-Trade B2B project is financed jointly by the National Technology Agency of Finland, the three Finnish companies participating in it, and VTT Information Technology. The project aims at automating inter-enterprise business processes where lots of simple activities, which could be easily automated, are still performed manually. The goal of the project is to work out a prototype system where each party in, for example, an ordering business process between enterprises, is represented by a software agent which coordinates business activities of the party by exchanging messages in an agent communication language with the agents representing the other parties involved.

At achieving its goal, the project builds on the present thesis by applying the methodology of the Business Agents' Approach to agent-oriented business modelling of the problem domain at hand and for designing an agent-based system for business process automation. The automation system built in the project directly utilizes activity diagrams of the extended AORML. The principle of automation in the project is very straightforward: *all the functionality described by the behaviour model of a business process type is to be performed by a software agent, with the exception of situations where the intervention by a human agent is absolutely necessary*. For example, in a business process of the type "Quoting" described by Figure 5-1, a human agent of the type Clerk approves each Quote to be sent to a Buyer, even though in principle software agents of sellers and buyers could handle quoting all by themselves. Intervention by a human agent is needed mainly for administrative or legal reasons, but also in cases where the criteria for selecting e.g. a supplier are not clear-cut.

In the Plug-and-Trade B2B project, executable models of business process types in the extended AORML are transformed into equivalent XML-based representations in order to enable the execution of the models by software agents and to grant that all the parties in a business process use the descriptions of the same business process type. To enable generation of XML-based representations of business process models, we have developed the corresponding XML Schema [XMLS] whose instances describe business process types in a machine-interpretable way. By using the schema, it is possible to represent business process types from different perspectives. For example, the models of the business process type "Quoting" are transformed into two XML-based representations that describe the business process type "Quoting" from the perspectives of a Seller and Buyer, respectively. There are plans of proposing the schema developed in the project to be adopted as a part of the RuleML [RuleML] standard draft. An excerpt of the schema is presented below:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="businessProcess" type="businessProcessType"/>
    <xs:complexType name="businessProcessType">
        <xs:sequence>
            <xs:element name="perspective" type="nameType"/>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="elementaryReactionRule" type="elementaryReactionRuleType"/>
                <xs:element name="forEachReactionRule" type="forEachReactionRuleType"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="reactionRuleType">
        <xs:all>
            <xs:element name="eventPart" type="eventPartType"/>
            <xs:element name="ruleName" type="nameType" minOccurs="0"/>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="elementaryReactionRuleType">
        <xs:complexContent>
            <xs:extension base="reactionRuleType">
                <xs:sequence maxOccurs="5">
                    <xs:element name="conditionPart" type="conditionPartType" minOccurs="0"/>
                    <xs:element name="mainActionPart" type="actionPartType" minOccurs="0"/>
                    <xs:element name="mainEffectPart" type="mentalEffectPartType" minOccurs="0"/>
                    <xs:element name="elseActionPart" type="actionPartType" minOccurs="0"/>
                    <xs:element name="elseEffectPart" type="mentalEffectPartType" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="forEachReactionRuleType">
        <xs:complexContent>
            <xs:extension base="reactionRuleType">
                <xs:all>
                    <xs:element name="actionPart" type="actionPartType"/>
                    <xs:element name="conditionPart" type="conditionPartType"/>
                </xs:all>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
...
</xs:schema>
```

Each party in a business process is represented by a software agent. In a preparatory stage, the agent reads the XML-based descriptions of the business process types that the party is involved in, and creates the corresponding internal executable representations of these business process types. After that, the descriptions of the business process types are ready to be interpreted by the agent in the course of process instances. A business process instance is started in response to the corresponding request by an internal human agent or in reaction to receiving the matching message from a software agent representing some other party. For example, an instance of the quoting business process at the buyer's side is triggered by the buyer's internal human agent of the type Clerk, while the same business process at the seller's side is started by receiving from the buyer a message of the request inform(?Quote) type. The software agent communicates with human agents (e.g., clerks) via a graphical user interface, while the messages exchanged between software agents are represented in the FIPA ACL [ACL97].

An agent representing a company needs to interact with the internal information systems, e.g. with Enterprise Resource Planning (ERP)- or Enterprise Application Integration (EAI)-systems, of the company. Because of the heterogeneity of such systems, their modelling is not as straightforward as that of business processes. In fact, the corresponding techniques and tools do not yet exist. However, the extended AORML enables to describe interfaces to the internal systems of a company at a high level of abstraction. For example, the interface to the product database of a Seller is represented as the agent's internal object :ProductDatabase shown in Figure 5-1 which includes the internal object type ProductItem. The instances of the latter represent types of product items that the company sells. Each instance of ProductItem is characterized by a number of attributes, like productID, productName, and itemsAvailable, and the intensional predicate isAvailable(Integer). At the implementation level to this

predicate corresponds a method in Java [JAVA] with the signature isAvailable(int quantity) : boolean. When the method is invoked by the agent, firstly the description of the corresponding product item is retrieved from the product database by using the following SQL query where value stands for the identifier of the product item:

```
select * from products where PRODUCT_ID = 'value'
```

After the attribute values of the product item retrieved have been copied into the respective attributes unitPrice and itemsAvailable of the given instance of ProductItem, the method isAvailable calculates the availability of the product item based on the values of the product item's attribute itemsAvailable and the method's formal parameter quantity, and returns.

For graphical modelling of business process types, Integrated Business Process Editor has been created based on the CONE (COnceptual NEtwork) Software worked out at VTT Information Technology. The Integrated Business Process Editor also enables to transform graphical descriptions of business process types in the extended AORML into their XML-based representations. An agent-based prototype system consisting of the software agents representing the Seller and Buyer has been implemented by using the FIPA-based JADE [JADE] agent platform. At both sides, the system consists of the Business Process Interpreter and JADE agent that invoke each other.

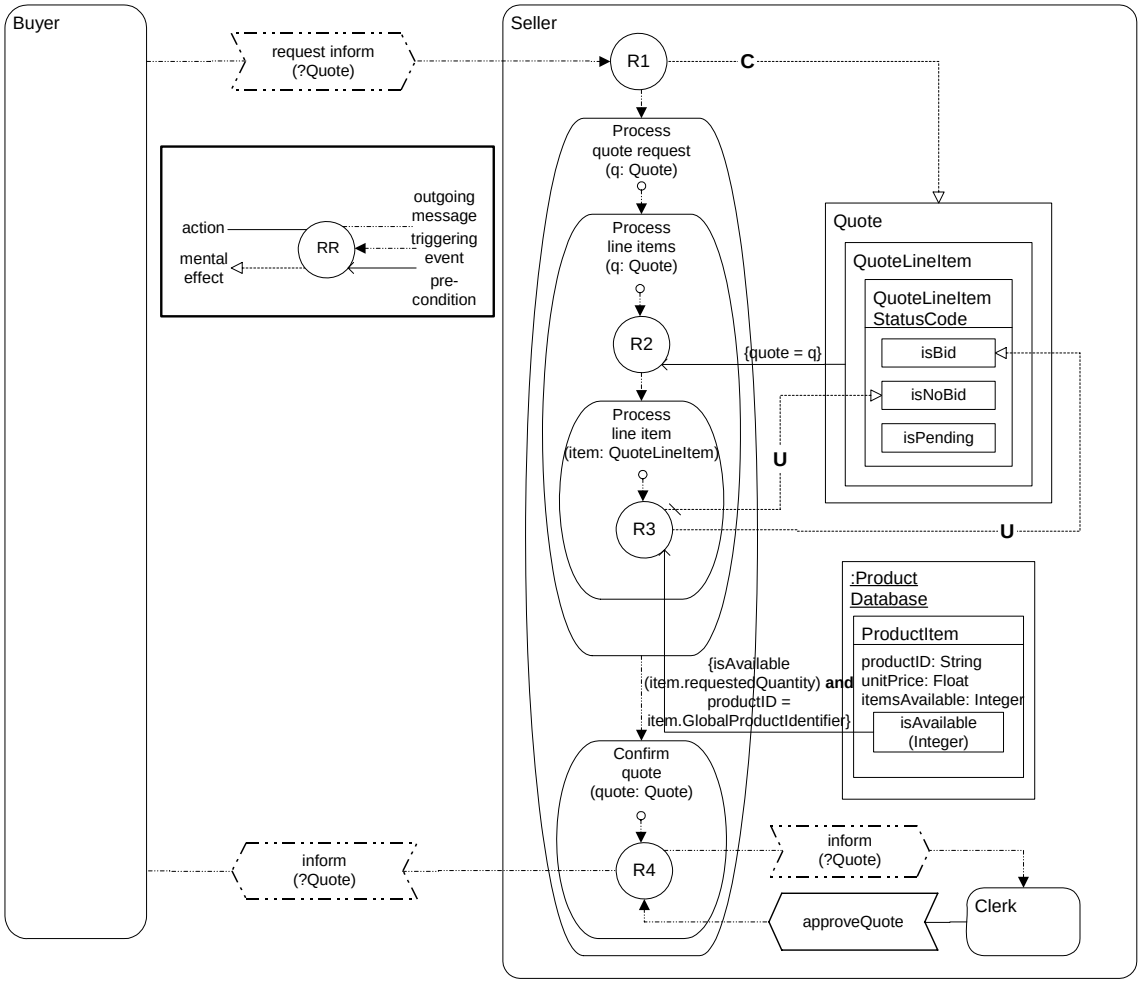


Figure 5-1. The business process type "Quoting" with the Seller in focus.

5.6. FUTURE RESEARCH WORK AND APPLICATION AREAS

The methodology presented in this thesis is just the first step towards a more flexible modelling approach and simulation system with a looser integration between models of business process types and actual business process instances carried out between agents. In order to achieve this, agents should be able to reason about their actions. This would enable a human or an automated (software) agent to select at each step of a business process from many alternatives the most appropriate actions to be performed. As it was pointed out in section 5.4, the first step towards this, which is allowing a reaction rule to query an agent's activity state at run time, has already been achieved. Such an approach would require two kinds of rules: *control rules* used in behavioural constructs and *meta-level rules* which would be used for selecting control rules to be applied based on the agent's information and activity states. In the present approach, these two kinds of rules are united. The resulting system would resemble the Meta-Level Architecture proposed in [Genesereth83] where actions of an agent include a set of base-level actions and a set of meta-level axioms that constrain how these actions are to be used.

An interesting research topic would be relating our modelling methodology to Web Services. Before getting to this, the difference between an approach based on Web Services and an agent-oriented approach like ours should be clarified. Firstly, even though Web Services Description Language (WSDL) [WSDL] can represent message sequences consisting of e.g. receiving a message and replying to it, this is far from the dynamics of agent communication protocols like FIPA Contract Net [FIPA]. Secondly, in principle an interface to an agent can be modelled and implemented as an interface based on Web Services, even though its feasibility still needs to be studied. We should thus investigate, could the communication between agents based on speech acts be substituted in our models with the parties' calls of each other's Web Services? What would be the added value of such an approach?

We should continue incorporating the extended AORML into business and ontology modelling tools. As the first step, the support for the extended AORML within the CONE (Conceptual Network) Software, which was mentioned in section 5.5, should be further enlarged. Secondly, one or more other, possibly open domain, modelling tools should be complemented with the support for the extended AORML.

We should also investigate more thoroughly the relationship between our approach and Model Driven Architecture (MDA) [MDA] of OMG, especially with regard to the simulation of business process models on JADE. In connection with the latter, MDA was briefly described in section 3.8.6. As it is reported in [MDASurvey], 75 percent of the companies surveyed by Compuware are currently assessing MDA and 50 percent of them are planning to start implementation of MDA in their organization within the next 12 months.

Finally, it would be interesting to compare our approach with the new coming UML 2.0 standard [OMG03b] which is claimed to be suitable for business process analysis [Adhikari03].

Some of the other new questions and research challenges that arise from our approach are:

- How to incorporate into our approach the modelling of goal-oriented *proactive behaviour* based on planning and plan execution.
- How can commitments/claims be used in real agent-oriented information systems? What is their operational semantics?
- How can we handle more systematically exceptions (now they are handled mainly by the ELSE-constructs of reaction rules) to standard processes (for instance, when a customer does not appear to pick up a car as agreed or when a response to a quote has not arrived in due time)? Possibly as violations of commitments?
- How can we introduce the feature of learning by an agent based on instances of action events and commitments/claims stored in the agent's VKB?
- How can we more precisely model interfaces to human agents (now they are modelled by using non-communicative action event types)?

The following areas of application can be envisioned for the Business Agents' Approach:

- Precise business modelling.
- Simulation of business and manufacturing processes.
- Business and manufacturing process automation.

References

[Aalst00] Aalst, W. M. P. Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets. *Systems Analysis - Modelling - Simulation*, 34(3):335-367, 1999.

[Aalst03a] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., Barros, A. P. Workflow Patterns. *Distributed and Parallel Databases,* 14(3), pp. 5-51, July 2003.

[Aalst03b] van der Aalst, W. M. P. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, Jan/Feb 2003.

[ACL97] *Agent Communication Language*, FIPA 97 Specification, http://www.fipa.org

[Adhikari03] Adhikari, R. Is UML heading for fragmentation? *Application Development Trends*, October 2003, Vol. 10, No. 10.

[AMICE93] ESPRIT Consortium AMICE, editor. *CIMOSA – Open System Architecture for CIM*. 2nd revised and expanded edition, Springer-Verlag, Berlin, 1993.

[Antikainen01] Antikainen, H., Bäck, A. *Challenges of electronic advertising processes in newspapers*. IFRA Special Report. VTT Information Technology, 2001.

[AOIS00] Wagner, G., Lesperance, Y., Yu, E. (eds.). *Agent-Oriented Information Systems 2000*. Proceedings of the 2nd International Workshop at CAiSE*00, Stockholm, June 2000. iCue Publishing, Berlin, 2000.

[Austin62] Austin, J. *How To Do Thing with Words*. Urmson Editor, Clarenson Press, UK, 1962.

[Balzer00] Balzer, W., Tuomela, R. Social Institutions, Norms, and Practices. *Proceedings of the Workshop On Norms and Institutions in Multi-Agent Systems at the Fourth International Conference on Autonomous Agents*. Barcelona, Spain, June 4, 2000.

[Barbuceanu99] Barbuceanu, M., Gray, T., Mankovksi, S. Roles of Obligations in Multiagent Coordination, *Applied Artificial Intelligence*, 13(1), 11–38, 1999.

[Bellifemine01] Bellifemine, F., Poggi, A., Rimassa, G. Developing multi-agent systems with a FIPA-compliant agent framework. *Software – Practice and Experience* 31 (2001) 103-128.

[Berio99] Berio, G., Vernadat, F. B. New developments in enterprise modelling using CIMOSA. *Computers in Industry* 40(1999), pp. 99-114.

[Berndtsson97] Berndtsson, M., Chakravarthy, S., Lings, B. Task Sharing Among Agents Using Reactive Rules. *Proceedings of the Second IFCIS Conference on Cooperative Information Systems* (CoopIS'97), Charlton, South Carolina, June 1997.

[Blanchard95] Blanchard, T. Meta Model Elements as a Foundation for Implementation of Business Rules. *Proceedings of the OOPSLA'95 Workshop on Metamodelling in OO*, October 15, 1995, http://saturne.info.uqam.ca/Labo_Recherche/Larc/MetamodellingWorkshop/Blanchard

[BPEL] *Business Process Execution Language for Web Services version 1.1*, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[BPML] *Business Process Modelling Language 1.0 and Business Process Modelling Notation 0.9*, http://www.bpmi.org/

[BR00] *Defining Business Rules - What Are They Really?* The Business Rules Group, formerly known as the GUIDE Business Rules Project, Final Report, revision 1.3, July, 2000. Prepared by D. Hay and K. A. Healy, http://businessrulesgroup.org/first_paper/br01c0.htm

[Brayshaw91] Brayshaw, M., Eisenstadt, M. A practical graphical tracer for Prolog. *International Journal of Man-Machine Studies*, 35(5):597–631, 1991.

[Brazier97] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., Treur, J. DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. *International Journal of Cooperative Information Systems*, 6(1), 67-94, 1997.

[Bubenko01] Bubenko, J. A. jr, Brash, D., Stirna, J. *EKD User Guide*. Kista, Dept. of Computer and Systems Science, Royal Institute of Technology (KTH) and Stockholm University, Stockholm, Sweden, 2001, http://www.dsv.su.se/~js/ekd_user_guide.html

[Bubenko93] Bubenko, J. A. Extending the Scope of Information Modelling. *Proceedings of the 4th International Workshop on the Deductive Approach to Information Systems and Databases*, Costa Brava, Catalonia, September 20-22, 1993, pp. 73-98.

[Bubenko94] Bubenko, J. A., jr, Kirikova, M. "Worlds" in Requirements Acquisition and Modelling. In: Kangassalo, H., Wangler, B. (eds.), *Proceedings of the 4th European - Japanese Seminar on Information Modelling and Knowledge Bases*, 31 May - 3 June 1994, Stockholm, Sweden. IOS Press, Amsterdam, 1994.

[Burmeister98] Burmeister, B., Bussmann, S., Haddadi, A., Sundermeyer, K. Agent-Oriented Techniques for Traffic and Manufacturing Applications: Progress Report. In: Jennings, N. R., Wooldridge, M. J. (eds.), *Agent Technology: Foundations, Applications, and Markets*. Springer, 1998.

[Bussler94] Bussler, C., Jablonski, S. Implementing Agent Coordination for Workflow Management Systems Using Active Database Systems. *Proceedings of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems* (RIDE-ADS'94), Houston, Texas, USA, February 1994.

[Cockburn97a] Cockburn, A. Goals and Use Cases. *Journal of Object-Oriented Programming*, September 1997.

[Cockburn97b] Cockburn, A. Using Goal-based Use Cases. *Journal of Object-Oriented Programming*, November/December 1997.

[Cockburn01] Cockburn, A. *Writing Effective Use Cases*. Addison-Wesley, 2001.

[Corcho03] Corcho, O., Fernandez-Lopez, M., Gomez-Perez, A. Methodologies,tools and languages for building ontologies. Where is their meeting point? *Data &Knowledge Engineering* 46 (2003), 41 – 64.

[Curtis92] Curtis, W., Kellner, M. I., Over, J. Process Modelling. *Communications of the ACM*, 35(9), 1992, pp. 75-90.

[Davenport93] Davenport, T. H. *Process Innovation: Reengineering Work through Information Technology*. Harvard Business School Press, 1993.

[Dayal88] Dayal, U. Active Database Management Systems. *Proceedings of the 3rd International Conference on Data and Knowledge Bases*, San Metheo, 1988, pp. 150-169.

[DeMichelis97] De Michelis, G., Dubois, E., Jarke, M., et al. *Cooperative Information Systems: A Manifesto*, http://www.sts.tu-harburg.de/projects/EUCAN/manifesto.html

[Dewar91] Dewar, A. D., Cleary, J. G. Graphical display of complex information within a Prolog debugger. *International Journal of Man-Machine Studies*, 25(5):503–521, 1991.

[Dignum95] Dignum, F., Weigand, H. Modellling communication between cooperative systems. In: Lyytinen, K., Iivari, J., Rossi, M. (eds.), *Advanced Information Systems Engineering (LNCS-932)*, pages 140-153. Springer-Verlag, Berlin, 1995.

[Dignum97] Dignum, F., Kuiper, R. Combining Dynamic Deontic Logic and Temporal Logic for the Specification of Deadlines. *Proceedings of the 30th Hawaiian International Conference on Systems*, Wailea, Hawaii, 1997.

[Dignum99] Dignum, F. Autonomous agents with norms. *Artificial Intelligence and Law*, volume 7, 1999, pp. 69-79.

[Dignum02] Dignum, V., Dignum, F. Towards an agent-based infrastructure to support virtual organisations. In: Camarinha-Matos, L. M. (ed.), *Proceedings of PRO-VE'02 - IFIP Int. Conf. on Infrastructures for Virtual Enterprises*, 1-3 May 2002, Sesimbra, Portugal. Kluwer Academic Publishers, 2002.

[ebXML] *ebXML* (Electronic Business using eXtensible Markup Language), http://www.ebxml.org/

[EDO99] *The UML profile for enterprise distributed object computing*. Technical Report, Cooperative Research Centre for Enterprise Distributed Systems Technology (DSTC) and the University of Newcastle, 1999.

[Elammari99] Elammari, M., Lalonde, W. An Agent-Oriented Methodology: High-Level and Intermediate Models. *Proceedings of the 1st International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS'99)*. Seattle, USA, 1 May 1999 and Heidelberg, Germany, 14 – 15 June 1999. Seattle, Heidelberg, 1999.

[Eriksson99] Eriksson, H.-E., Penker, M. Business Modelling with UML. *Rose Architect*, Fall 1999.

[Eriksson00] Eriksson, H.-E., Penker, M. *Business Modelling with UML: Business Patterns at Work*. John Wiley & Sons, Inc., 2000.

[Eshuis02a] Eshuis, R. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. Ph.D. thesis, CTIT Ph.D.-thesis Series No. 02-44 Centre for Telematics and Information Technology (CTIT), University of Twente, The Netherlands, 2002.

[Eshuis02b] Eshuis, R., Jansen, D. N., Wieringa, R. J. Requirements-level semantics and model checking of object-oriented statecharts. *Requirements Engineering Journal*, 7(4):243-263, 2002.

[Farhoodi96] Farhoodi, F., Graham, I. A Practical Approach to Designing and Building Intelligent Software Agents. *Proceedings of the First International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology* (PAAM'96), London, UK, April 1996, pp. 181-204.

[Fikes71] Fikes, R. E. , Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4): 189-208, 1971.

[FIPA] *Foundation for Intelligent Physical Agents* (FIPA), http://www.fipa.org

[Gates03] Gates, L. Data modelers, BPM gradually unite. *Application Development Trends*, February 2003, Volume 10, Number 2.

[Genesereth83] Genesereth, M. An Overview of Meta-Level Architecture. *Proceedings of the Third National Conference on Artificial Intelligence* (AAAI-83). Menlo Park, Calif., AAAI Press, 1983, 119-124.

[Genesereth94] Genesereth, M. R., Ketchpel, S. P. Software agents. *Communication of the ACM*, 37(7):48–53, 1994.

[Gottesdiener99] Gottesdiener, E. Business rules as requirements. *Software Development*, 7(12), December 1999.

[Gray93] Gray, J., Reuter, A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, 1993.

[Gruber93] Gruber, T. R. A Translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5(2), 199-220, 1993, http://ksl-web.stanford.edu/knowledge-sharing/papers/README.html#ontolingua-intro

[Halpin96] Halpin, T. Business Rules and Object Role Modelling. *Database Programming and Design*, October 1996.

[Hammer93] Hammer, M., Champy, J. *Reengineering the Corporation*. New York, Harper Collins, 1993.

[Harel87] Harel, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Hay97] Hay, D. C. The Zachman Framework: Introduction. *The Data Administration Newsletter*, Issue 1 (Summer, 1997), http://www.tdan.com

[Herbst95] Herbst, H. A Meta-Model for Specifying Business Rules in Systems Analysis. In: Iivari, J., Lyytinen, K., Rossi, M. (eds.), *Proceedings of the Seventh Conference on Advanced Information Systems Engineering* (CAiSE'95). Springer, 1995, pp. 186 – 199.

[Herbst97] Herbst, H. *Business Rule-Oriented Conceptual Modelling* (Contributions to Management Science). Springer-Verlag, 1997.

[Høydalsvik93] Høydalsvik, G. M., Sindre, G. On the Purpose of Object-Oriented Analysis. OOPSLA'93 Conference Proceedings, *ACM Sigplan Notices*, October 1993, pp. 240-253.

[Hurlbut98] Hurlbut, R.. *Managing Domain Architecture Evolution Through Adaptive Use Case and Business Rule Models*. PhD thesis, Illinois Institute of Technology, USA, 1998.

[IDEF] IDEF Family of Methods – A structured approach to enterprise modelling and analysis, http://www.idef.com

[IDEF94] *Information Integration for Concurrent Engineering (IICE) IDEF5 Method Report*. Prepared by Knowledge Based Systems, Inc., 1994, http://www.idef.com/idef5.html

[Jacobson92] Jacobson, I., et al. *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Addison-Wesley, Reading, MA, 1992.

[JADE] *JADE Programmer's Guide*, http://jade.cselt.it/

[JAVA] The Source for Java Developers, http://www.java.sun.com

[Jennings98] Jennings, N. R., Sycara, K., Wooldridge, M. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1), 7-38, 1998.

[Jennings00] Jennings, N. R. On agent-based software engineering. *Artificial Intelligence* 117(2000), pp. 277-296.

[Kappel98] Kappel, G., Rausch-Schott, S., Retschitzegger, W. *Coordination in Workflow Management Systems – A Rule-Based Approach*. In: Conen, W., Neumann, G. (eds.), Coordination Technology for Collaborative Applications – Organizations, Processes, and Agents, Springer LNCS 1364, pp. 99-120, 1998.

[Karageorgos02] Karageorgos, A., Mehandjiev, N., Thompson, S. RAMASD: a semi-automatic method for designing agent organisations. *The Knowledge Engineering Review*, Vol. 17:4, 331-358.

[Kardasis03] Kardasis, P., Loucopoulos, P. Managing Business Rules during the Requirements Engineering Process in Rule-Intensive IT Projects. In: Abramowicz, W., Klein, G. (eds.), *Proceedings of the 6th International Conference on Business Information Systems* (BIS 2003), Colorado Springs, Colorado, USA, 4-6 June, 2003.

[Kavakli98] Kavakli, V., Loucopoulos, P. Goal-Driven Business Process Analysis Application in Electricity Deregulation. *Proceedings of the 10th International Conference on Advanced Information Systems Engineering* (CAiSE'98). Springer, 1998.

[Kendall96] Kendall, E. A., Malkoun, M. T. , Jiang, C. A Methodology for Developing Agent Based Systems for Enterprise Integration. In: Lukose, D., Zahng, C. (eds.), *Proceedings of the First Australian Workshop on Distributed Artificial Intelligence*. Springer-Verlag, 1996.

[Kieser92] Kieser, A., Kubicek, H. *Organisation*. 3rd Edition, Berling/New York: De Gruyter, 1992.

[Kirikova00] Kirikova, M. Explanatory capability of enterprise models. *Data & Knowledge Engineering*, 33 (2000), pp. 119-136.

[KlasseObjecten] An introduction to MDA, http://www.klasse.nl/english/mda/mda-introduction.html

[KQML] *Knowledge Query and Manipulation Language* (KQML), http://www.cs.umbc.edu/kqml/

[Lesperance99] Lespérance, Y., Kelley, T. G., Mylopoulos, J., Yu, E. S. K. Modelling Dynamic Domains with ConGolog. *Proceedings of the 11th International Conference on Advanced Information Systems Engineering* (CAiSE-99). LNCS Vol. 1626, Springer-Verlag, Berlin, pp. 365-380.

[Loucopoulos91] Loucopoulos, P. Theodoulidis, B., Pantazis, D. Business Rules Modelling: Conceptual Modelling and Object Oriented Specifications. Object Oriented Approach. In: *Information Systems, Proceedings of the IFIP TC8/WG8.1 Working Conference*, Netherlands, 28-31 Nov. 1991, pp. 323-342, 1991.

[Lubell02] Lubell, J. *XML Representation of Process Descriptions (PSL-XML)*. National Institute of Standards and Technology (NIST), Manufacturing Systems Integration Division, 2002, http://ats.nist.gov/psl/xml/process-descriptions.html

[Manna92] Manna, Z., Pnueli, A. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[Martin98] James Martin and James Odell. *Object-Oriented Methods: A Foundation* (UML Edition). Prentice-Hall, 1998.

[McCarthy82] McCarthy, W. E. The REA accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review*, LVII(3):554–578, July 1982.

[MDA] OMG Model Driven Architecture, http://www.omg.org/mda/

[MDASurvey] MDA starts to take off. *SIGS Application Development Advisor*, Nov/Dec 2003, Vol. 7, No. 6.

[MDC99] *Meta data coalition open information model, business engineering model, business rules*. Review draft. Kista, Dept. of Computer and Systems Science, Royal Institute of Technology (KTH) and Stockholm University, July 1999.

[Medina-Mora92] Medina-Mora, R., Winograd, T., Flores, R., Flores, C.-F. The Action Workflow Approach to Workflow Management Technology. *Proceedings of the 4th ACM Conference on CSCW*, Toronto, Canada, 31 October – 4 November 1992, pp. 281–288.

[Metsker97] Metsker, S. J. Thinking Over Objects. *Object Magazine*, May, 1997.

[Moriarty93] Moriarty, T. The Next Paradigm. *Database Programming and Design*, Vol. 6, No. 2, pp. 66-69, 1993.

[Mylopoulos01] Mylopoulos, J., Kolp, M., Castro, J. UML for Agent-Oriented Software Development: the Tropos Proposal. *Proceedings of the Fourth International Conference on the Unified Modelling Language*, Toronto, Canada, October 2001.

[Neufeld97] Neufeld, E., Kusalik, A., Dobrohoczki, M. Visual metaphors for understanding logic program execution. In: Davis, W., Mantel, M., Klassen, V. (eds.), *Graphics Interfaces*, pages 114–120, 1997.

[Nilsson98] Nilsson, A. G. *Perspectives on Business Modelling: Key Issues in Corporate, Organisational and Systems Development*. Keynote Address at the Eight Annual Workshop on Information Technologies and Systems (WITS'98), Helsinki, Finland, December 12 - 13, 1998.

[OBP00] *Organizing Business Plans*. The Standard Model for Business Rule Motivation. Prepared by the Business Rules Group, November 15, 2000, revision 1.0, www.businessrulesgroup.org

[Odell95] Odell, J. Meta-Modelling. *Proceedings of the OOPSLA'95 Workshop on Metamodelling in OO*, October 15, 1995, http://www.info.uqam.ca/Labo_Recherche/Larc/MetamodellingWorkshop/Odell/metamodelling/

[Odell99] Odell, J. A Flock is Not a Bird: Agents and Beyond. *Data To Knowledge Newsletter*, Vol. 27, No. 1, January / February 1999.

[Odell00]. Odell, J. H., Van Dyke Parunak, H., Bauer, B. Extending UML for Agents. *Proceedings of the Second International Bi-Conference Workshop on Agent-Oriented Information Systems* (AOIS-2000), 5-6 June 2000, Stockholm (Sweden) and 30 July 2000, Austin (Texas, USA).

[Oja01] Oja, M., Tamm, B., Taveter, K. Agent-based software design. *Proc. Estonian Acad. Sci. Eng*., 2001, 7, 1, 5-21.

[OMG92] Object Management Group. *Object-oriented analysis and design, reference model*. Discussion paper, November 1992, http://www.omg.org/

[OMG03a] *OMG Unified Modelling Language Specification*, Version 1.5, March 2003, http://www.uml.org/

[OMG03b] *Unified Modelling Language: Superstructure*. Version 2.0, August 2003, http://www.uml.org/

[OPEN] *Object-oriented Process, Environment, and Notation* (OPEN), http://www.open.org.au/

[Ould95] Ould, M. A. *Business Processes: Modelling and Analysis for Re-Engineering and Improvement*. John Wiley & Sons, 1995.

[Ow88] Ow, S. P., Smith, S. F., Howie, R. A. Cooperative Scheduling System. In: Oliff, M. D. (ed.), *Expert Systems and Intelligent Manufacturing*, Proceedings of the Second International Conference on Expert Systems and the Leading Edge in Production Planning and Control, May 3-5, 1988, Charleston, South Carolina. Elsevier Science Publishing Co., Inc., 1988.

[PapiNet] *PapiNet*, http://www.papinet.org/

[Patterns03] *Workflow Patterns*, http://tmitwww.tm.tue.nl/research/patterns/

[Pernice95] Pernice, A., Doare, H., Rienhoff, O. (eds.): *Healthcare Card Systems: EUROCARDS Concerted Action Results and Recommendations*. Technology and Informatics 22, Amsterdam; IOS Press 1995. Annex p. 185 ff.

[Presley97] Presley, A. R. *A Representation Method to Support Enterprise Engineering*. Ph.D. thesis, the University of Texas at Arlington, USA, May 1997.

[Reyneri99] Reyneri, C. Operational building blocks for business process modelling. *Computers in Industry* 40(1999), pp. 115-123.

[RosettaNet] *Rosetta Net*, http://www.rosettanet.org/

[Ross97] Ross, R. G. *The Business Rule Book: Classifying, Defining and Modelling Rules*. Second Edition, Boston, Massachusetts, Database Research Group, Inc., 1997.

[Ross03] Ross, R. G. *Principles of the Business Rule Approach*. Addison-Wesley, 2003.

[RuleML] *The Rule Markup Initiative*, http://www.ruleml.org

[Salminen95] Salminen, A. EDIFACT for Business Computers: Has it Succeeded? *StandardView* Vol. 3, No. 1, March/1995.

[Sandy99] Sandy, G. *Rules the Missing Link in Requirements Engineering*. Internal Working Paper, http://www.business.vu.edu.au/infosyspapers/docs/1999/Sandy.pdf

[Searle85] Searle, J., Vanderverken, D. *Foundations of the Illocutionary Logic*. Cambridge University Press, UK, 1985.

[Shoham93] Shoham, Y. Agent-Oriented Programming. *Artificial Intelligence*, 60(1), 51-92, 1993.

[Singh99] Singh, M. An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *Artificial Intelligence and Law* 7 (1999) 97-113.

[Singh00] Singh, M. P. Synthesizing Coordination Requirements for Heterogeneous Autonomous Agents. *Autonomous Agents and Multi-Agent Systems*, 3, 2000, pp. 107-132.

[Sladek96] Sladek, A., Wolski, A. Modelling inter-organizational workflows. *Proceedings of the International Symposium on Applied Corporate Computing* (ISACC'96), Monterrey, Mexico, 30 Oct. – 1 Nov. 1996, pp. 13 – 22.

[Smith89] Smith, S. *The OPIS Framework for Modelling Manufacturing Systems*. Tech. report CMU-RI-TR-89-30, Robotics Institute, Carnegie Mellon University, December, 1989.

[Smith90] Smith, S., Ow, P. S., Muscettola, N., Potvin, J. Y., Matthys, D. OPIS: An Opportunistic Factory Scheduling System. *Proceedings of the Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 268-274. May, 1990.

[Smith95] Smith, S. Reactive Scheduling Systems. In: Brown, D. E., Scherer, W. T. (eds.), *Intelligent Scheduling Systems*. Kluwer Press, 1995.

[Smith97] Smith, S. F., Becker, M. A. An Ontology for Constructing Scheduling Systems. *Working Notes of 1997 AAAI Symposium on Ontological Engineering*. AAAI Press, March, 1997.

[Smith03] Smith, H., Fingar, P. *Business Process Mangement: The Third Wave*. 1st ed., Meghan-Kiffer Press, Tampa, Florida, USA, 2003.

[Sowa92] Sowa, J. F., Zachman, J. A. Extending and formalizing the framework for information systems architecture. *IBM Systems Journal* 31 (3) (1992).

[Sowa00] Sowa, J. F. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole, 2000.

[Sterling86] Sterling, L., Shapiro, E. *The Art of Prolog: Advanced Programming Techniques*. The MIT Press, 1986.

[Tamm87] Tamm, B. G., Puusepp, R., Tavast, R. *Analiz i modelirovanije proizvodstvennyh sistem* (Analysis and Modelling of Production Systems). Finansy i Statistika, Moskva, 1987. In Russian.

[Tamm96] Tamm, B., Taveter, K. A List-based Virtual Machine for COBOL. *Software - Practice and Experience*, Vol. 26 (12) (December 1996).

[Taveter01c] Taveter, K., Wagner, G. Agent-Oriented Enterprise Modelling Based on Business Rules. In: Kunii, H. S., Jajodia, S., Sølvberg, A. (eds.): *Conceptual Modelling – ER 2001*, 20th International Conference on Conceptual Modelling, Yokohama, Japan, November 27-30, 2001, Proceedings. Lecture Notes in Computer Science 2224, Springer, 2001, 527-540.

[Taveter02a] Taveter, K., Wagner, G. A Multi-perspective Methodology for Modelling Inter-enterprise Business Processes. In: Arisawa, H., Kambayashi, Y., Kumar, V., Mayr, H.C., Hunt, I. (eds.): *Conceptual Modelling for New Information Systems Technologies*, ER 2001 Workshops HUMACS, DASWIS, ECOMO, and DAMA, Yokohama, Japan, November 27-30, 2001, Revised Papers. Lecture Notes in Computer Science 2465, Springer, 2002, 403 – 416.

[TKT] *Tallinna Keraamikatehas AS*, http://www.keraamikatehas.ee/

[UMM] *UN/CEFACT Modelling Methodology* (UMM), http://www.ebxml.org/

[Uschold98] Uschold, M., King, M., Moralee, S., Zorgios, Y. The Enterprise Ontology. *The Knowledge Engineering Review*, 13(1), 31-90, 1998.

[VanAssche88] Van Assche, F., Layzell, P. J., Loucopoulos, P., Speltincx, G. Information systems development: a rule-based approach, *Journal of Knowledge-Based Systems*, 1(4), 227-234, 1988.

[Vernadat98] Vernadat, F. B. The CIMOSA languages. In: *Handbook on Architectures of Information Systems*, Springer-Verlag, Berlin, 1998, pp. 243–263.

[Wagner96] Wagner, G. Vivid Agents - How they Deliberate, How They React, How They Are Verified, http://tmitwww.tm.tue.nl/staff/gwagner/VividAgents.pdf. Extended version of: Wagner, G. A Logical And Operational Model of Scalable Knowledge- and Perception-Based Agents. In: Van de Velde, W., Perram. J. W. (eds.), *Agents Breaking Away, Proceedings of MAAMAW'96*, Springer Lecture Notes in Artificial Intelligence 1038, 1996.

[Wagner98] Wagner, G. *Foundations of Knowledge Systems with Applications to Databases and Agents*. Volume 13 of Advances in Database Systems, Kluwer Academic Publishers, 1998.

[Wagner99] Wagner, G. Agent-Oriented Enterprise and Business Process Modelling. *Proceedings of the First International Workshop on Enterprise Management and Resource Planning Systems* (EMRPS'99), Venice, November 1999.

[Wagner00a] Wagner, G. Agent-Object-Relationship Modelling. *Proceedings of the Second International Symposium "From Agent Theory to Agent Implementation"* (AT2AI-2), Vienna, Austria, April 2000.

[Wagner00b] Wagner, G., Schroeder, M. Vivid Agents: Theory, Architecture, and Applications. *Journal of Applied Artificial Intelligence* 14:7 (2000).

[Wagner01] Wagner, G. Agent-Oriented Analysis and Design of Organizational Information Systems. In: Barzdins, J., Caplinskas, A. (eds.), *Databases and Information Systems*, 4th International Baltic Workshop, Baltic DB&IS, Selected Papers. Vilnius, Lithuania, 1 – 5 May 2000. Kluwer Academic Publishers, Dordrecht, 2000.

[Wagner02] Wagner, G. A UML Profile for External AOR Models. *Proceedings of Third International Workshop on Agent-Oriented Software Engineering* (AOSE-2002), held at Autonomous Agents & Multi-Agent Systems (AAMAS 2002), Palazzo Re Enzo, Bologna, Italy – July 15, 2002. LNAI 2585, Springer-Verlag, 2002.

[Wagner03a] Wagner, G. The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behaviour. *Information Systems* 28:5 (2003), http://tmitwww.tm.tue.nl/staff/gwagner/AORML/AOR.pdf

[Wagner03b] Wagner, G., Tulba, F. Agent-Oriented Modeling and Agent-Based Simulation. In: Giorgini, P., Henderson-Sellers, B. (eds.), *Conceptual Modeling for Novel Application Domains*. Volume 2814 of Lecture Notes in Computer Science, Springer-Verlag, 2003, pp. 205-216.

[Wang01] Wang, X., Lesperance, Y. Agent-Oriented Requirements Engineering Using ConGolog and *i**. In: Wagner, G., Karlapalem, K., Lesperance, Y., Yu., E. (eds.), *Agent-Oriented Information Systems 2001*, Proceedings of the Third International Bi-Conference Workshop AOIS-2001. iCue Publishing, Berlin, 2001.

[Weigand97] Weigand, H., Verharen, E., Dignum, F. Dynamic Business Models as a Basis for Interoperable Transaction Design. *Information Systems*, Vol. 22, No. 2/3, pp. 139-154, 1997.

[Wooldridge00] Wooldridge, M., Jennings, N. R., Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3, 285-312, 2000.

[WS] *Web Services Activity of W3C*, http://www.w3.org/2002/ws/

[WSDL] *Web Services Description Language (WSDL) Version 1.2*. W3C Working Draft 3 March 2003, http://www.w3.org/TR/2003/WD-wsdl12-20030303/

[XMLS] *XML Schema 1.0*, http://www.w3.org/XML/Schema.

[XPDL] *Workflow Process Definition Interface - XML Process Definition Language (XPDL), October 25, 2002, Version 1.0* , http://www.wfmc.org/standards/standards.htm

[Yourdon96] Yourdon, E., Whitehead, K., Thomann, J., Oppel, K., Nevermann, P. *Mainstream Objects: An Analysis and Design Approach for Business*. Yourdon Press, 1996.

[Yu95a] Yu, E. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, Department of Computer Science, University of Toronto, 1995.

[Yu95b] Yu, E. S. K., Mylopoulos, J.: From E-R to 'A-R' - Modelling Strategic Actor Relationships for Business Process Reengineering. *International Journal of Intelligent and Cooperative Information Systems* 2/3 (1995) 125-144.

[Zachman87] Zachman, J. A. A framework for information systems architecture. *IBM Systems Journal* 26 (3) (1987).

[Zambonelli01] Zambonelli, F., Jennings, N. R., Wooldridge, M. Organisational Abstractions for the Analysis and Design of Multi-Agent Systems. In: Ciancarini, P. and Wooldridge, M. (eds.), *Agent-Oriented Software Engineering*, LNCS 1957, pages 127-141. Springer-Verlag, 2001.

[Zave97a] Zave, P. Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, Vol. 29, No. 4, 1997, pp. 315-321.

[Zave97b] Zave, P., Jackson, M. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, 1997, pp. 1-30.

[Zeng99] Zeng, D. D., Sycara, K. Dynamic Supply Chain Structuring for Electronic Commerce Among Agents. In: Klusch, M. (ed.), *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Springer, 1999.

Appendix A. The grammar for the proposed modification of OCL

```
oclFile              := ( "package" packageName
                          oclExpressions
                          "endpackage"
                        )+
packageName          := pathName
oclExpressions       := ( constraint )*
constraint           := contextDeclaration
                        ( ( "def" defExpression )
                        |
                        ( stereotype name? ":" oclExpression )
                        )+
contextDeclaration   := "context"
                        ( operationContext | classifierContext  )
classifierContext    := nameList
nameList             := ( name (":" name)? )
                        ("," name (":" name)? )*
operationContext     := name "::" operationName
                        "(" formalParameterList ")"
                        ( ":" returnType )?
stereotype           := ( "pre" | "post" | "inv" )
operationName        := name | "=" | "+" | "-" | "<" | "<=" |
                        ">=" | ">" | "/" | "*" | "<>" |
                        "implies" | "not" | "or" | "xor" | "and" | “IF”
formalParameterList  := ( name ":" typeSpecifier
                        ("," name ":" typeSpecifier )*
                        )?
typeSpecifier        := simpleTypeSpecifier
                        | collectionType
collectionType       := collectionKind
                        "(" simpleTypeSpecifier ")"
oclExpression        := (letExpression* "in")? expression
returnType           := typeSpecifier
expression           := logicalExpression
letExpression        := "let" name
                        ( "(" formalParameterList ")" )?
                        ( ":" typeSpecifier )?
                        "=" expression
defExpression        := name
                        ( "(" formalParameterList ")" | ( ":" typeSpecifier ) )
                        ("=" expression)?
ifExpression         := "if" expression
                        "then" expression
                        "else" expression
                        "endif"
logicalExpression    := relationalExpression
                        ( logicalOperator
                        relationalExpression
                        )*
relationalExpression := additiveExpression
                        ( relationalOperator
                        additiveExpression
                        )?
```

Appendix A (continued). The grammar for the proposed modification of OCL

```
additiveExpression       := multiplicativeExpression
                            ( addOperator
                            multiplicativeExpression
                            )*
multiplicativeExpression := unaryExpression
                            ( multiplyOperator
                            unaryExpression
                            )*
unaryExpression          := ( unaryOperator
                              postfixExpression
                            )
                            | postfixExpression
postfixExpression        := primaryExpression
                            ( ("." | "->")propertyCall )*
primaryExpression        := literalCollection
                            | literal
                            | propertyCall
                            | "(" expression ")"
                            | ifExpression
propertyCallParameters   := "(" ( declarator )?
                            ( actualParameterList )? ")"
literal                  := string
                            | number
                            | enumLiteral
enumLiteral              := name "::" name ( "::" name )*
simpleTypeSpecifier      := pathName
literalCollection        := collectionKind "{"
                            ( collectionItem
                            ("," collectionItem )*
                            )?
                            "}"
collectionItem           := expression (".." expression )?
propertyCall             := pathName
                            ( timeExpression )?
                            ( qualifiers )?
                            ( propertyCallParameters )?
qualifiers               := "[" actualParameterList "]"
declarator               := name ( "," name )*
                            ( ":" simpleTypeSpecifier )?
                            ( ";" name ":" typeSpecifier "="
                            expression
                            )?
                            "|"
pathName                 := name ( "::" name )*
timeExpression           := "@" "pre"
actualParameterList      := expression ("," expression)*
logicalOperator          := "and" | "or" | "xor" | "implies" | "IF"
collectionKind           := "Set" | "Bag" | "Sequence" | "Collection"
relationalOperator       := "=" | ">" | "<" | ">=" | "<=" | "<>"
addOperator              := "+" | "-"
multiplyOperator         := "*" | "/"
unaryOperator            := "-" | "not"
typeName                 := charForNameTop charForName*
name                     := charForNameTop charForName*
charForNameTop           := /* Characters except inhibitedChar and ["0"-"9"]; the available
                            characters shall be determined by the tool implementers ultimately.*/
charForName              := /* Characters except inhibitedChar; the available
                            characters shall be determined by the tool implementers ultimately.*/
```

Appendix A (continued). The grammar for the proposed modification of OCL

```
inhibitedChar   := /* The available inhibited characters shall be determined by the tool
                   implementers ultimately.*/
number          := ["0"-"9"] (["0"-"9"])*
                   ( "." ["0"-"9"] (["0"-"9"])* )?
                   ( ("e" | "E") ( "+" | "-" )? ["0"-"9"] (["0"-"9"])*
                   )?
string          := "'"
                   (( ~["'","\\","\n","\r"] )
                   | ("\\"
                   ( ["n","t","b","r","f","\\","'","\""]
                   | ["0"-"7"]
                   ( ["0"-"7"] ( ["0"-"7"] )? )?
                   )
                   )
                   )*
                   "'"
```

businessProcessType := (reactionRule)+
reactionRule := "CONTEXT" *classifierContext+ defExpression**
"ON" eventPart (preconditionPart)?
"THEN" (actionPart | postconditionPart)+
("ELSE" (actionPart | postconditionPart)+)?
eventPart := eventExpression (("AND" | "OR" | "XOR") eventExpression)*
preconditionPart := "IF" *expression*
actionPart := immediateActionPart | deferredActionPart
immediateActionPart := actionExpression (& actionExpression)*
deferredActionPart := ("OR" | "XOR")? (reactionRule)+
postconditionPart := "EFFECT" *expression*
eventExpression := "RECEIVE MESSAGE" messageTemplate
"FROM" agentID |
"PERCEIVE ACTION EVENT" actionTemplate
"CREATED BY" agentID |
"PERCEIVE NON-ACTION EVENT" actionTemplate |
"END" activityID |
"START" activityTemplate
actionExpression := "SEND MESSAGE" messageReference "TO" agentID |
"PERFORM ACTION" actionReference "FOR" agentID |
"START ACTIVITY" activityReference (*number* TIMES)?
"CANCEL" activityType |
"CANCEL PROCESS"
messageTemplate := performativeName contentTemplate?
contentTemplate := "(" term1 ")"
term1 := actionTemplate | *formalParameterList*
actionTemplate := actionID | actionID "(" term1 ")"
performativeName := "accept-proposal" | "agree" | "cancel" | "cfp" | "confirm" | "disconfirm" |
"failure" | "inform" | "not-understood" | "propose" | "query-if" | "refuse" |
"reject-proposal" | "request"
activityTemplate := activityID | activityID "(" term2 ")"
term2 := *formalParameterList*
messageReference := performativeName contentExpression?
contentExpression := "(" term3 ")"
term3 := actionReference | *actualParameterList* | "ACHIEVE" *expression*
actionReference := actionID | actionID "(" term3 ")"
activityReference := activityType | activityType "(" *actualParameterList* ")"
actionID := *name*
activityID := *name*
agentID := *name*

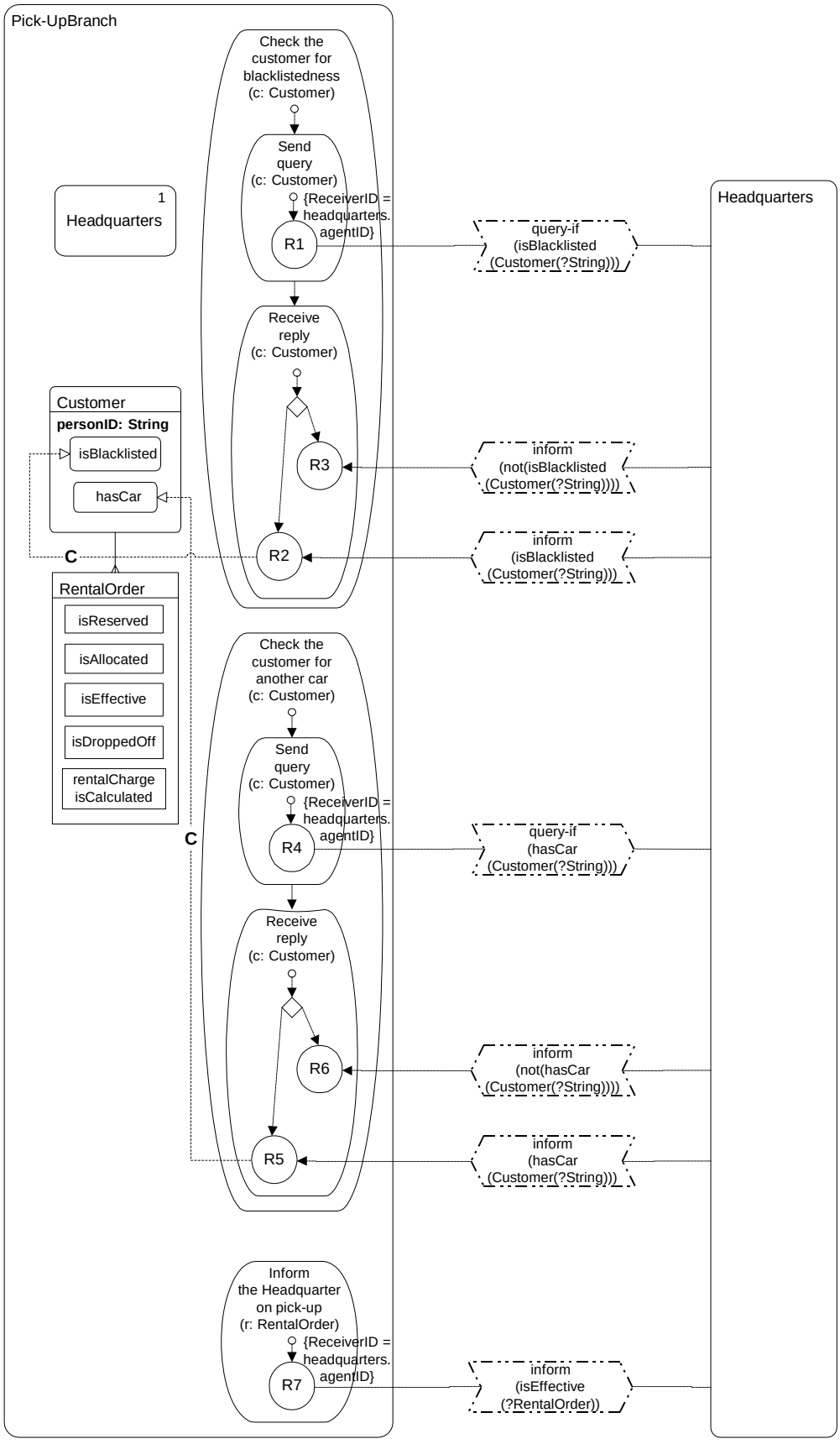
```
context CarGroup::hasCapacity(pt: Date, dt: Date): Boolean
post: result = (self.rentalCar->exists(not(isScheduledForService) and not(hasOverlappingRentalOrder(pt, dt))))

context RentalOrder
def: overlaps (pt: Date, dt: Date): Boolean = (dt >= pickUpTime and dt <= dropOffTime) or
(pt >= pickUpTime and pt <= dropOffTime) or (pt <= pickUpTime and dt >= dropOffTime) or
(pt >= pickUpTime and dt <= dropOffTime)

context RentalCar
def: isSchedulable (r : RentalOrder): Boolean = (serviceStartTime <= r.pickUpTime and serviceEndTime <= r.pickUpTime)
or (serviceStartTime >= r.dropOffTime and serviceEndTime >= r.dropOffTime)
def: hasOverlappingRentalOrder (pt: Date, dt: Date) : Boolean = branch.rentalOrder->exists(overlaps(pt, dt))

context RentalCar inv:
self.isAvailable IF
self.isPresent and
self.rentalOrder->isEmpty() and
not self.requiresService and
not self.isScheduledForService

context RentalCar inv:
self.isAvailableWithMinMileage IF
self.isAvailable and self.carGroup.rentalCar->select(isAvailable)->forAll(self.mileage <= mileage)

context RentalCar::isAvailableOfOwnGroup(r: RentalOrder): Boolean
post: result = (carGroup = r.carGroup and isAvailableWithMinMileage)

context RentalCar::isAvailableOfNextHigherGroup(r: RentalOrder): Boolean
post: result = (carGroup.nextLowerGroup->notEmpty() and
carGroup.nextLowerGroup = r.carGroup) and isAvailableWithMinMileage)

context RentalCar::isAvailableWithBumpedUpgrade(r : RentalOrder): Boolean
post: result = (carGroup.nextLowerGroup->notEmpty() and
carGroup.nextLowerGroup.nextLowerGroup->notEmpty() and
carGroup.nextLowerGroup.nextLowerGroup = r.carGroup) and isAvailableWithMinMileage)

context RentalOrder::canBeReAllocated(r: RentalOrder): Boolean
post: result =(self = RentalOrder->any(isAllocated and carGroup = r.nextHigherCarGroup and
pickUpTime >= r.pickUpTime and dropOffTime <= r.dropOffTime))

context RentalCar::isAvailableOfNextLowerGroup(r: RentalOrder): Boolean
post: result =  (carGroup.nextHigherGroup->notEmpty() and
carGroup.nextHigherGroup = r.carGroup) and isAvailableWithMinMileage)

context RentalCar::isAvailableNotPresent(r: RentalOrder): Boolean
post: result = (r.carGroup.rentalCar->select(isPickedUp and
not rentalOrder.overlaps(r.pickUpTime,  r.dropOffTime))->forAll(self.mileage <= mileage))

context Proposal inv:
self.isCheapest IF
branch.proposal->forAll(self.priceForTransfer <= priceForTransfer)

context Customer inv:
self.isQualifiedForRental IF
self.age >= 25

self.hasCar IF
self.rentalOrder->notEmpty() and self.rentalOrder->exists(isEffective)
```

```
context UnitCapacityInterval inv:
availableDuration = 480 * unitCapacityResource.capacity.numberOfResources –
productionActivity.getDuration(startTime, endTime)->sum()

context UnitCapacityInterval
def: capacityPerHour (a : ProductionActivity) : Real = a.productionActivityType.numberOfProductsPerHour *
unitCapacityResource.capacity.numberOfResources
def: requiredDuration (a : ProductionActivity) : Integer = (a.quantity / capacityPerHour(a)) * 60

context UnitCapacityInterval::isSchedulable (a : ProductionActivity): Boolean =
a.earliestStartTime <= endTime and requiredDuration(a) <= availableDuration and
self.unitCapacityResource.unitCapacityInterval->select(a.earliestStartTime <= endTime and
requiredDuration(a) <= availableDuration)->forAll(self.startTime <= startTime)

context BatchCapacityInterval inv:
availableCapacity = batchCapacityResource.capacity.numberOfResources * batchCapacityResource.capacity.batchSize –
productionActivity.quantity->sum()

context BatchCapacityInterval::isSchedulable (a : ProductionActivity) : Boolean =
a.earliestStartTime <= startTime and a.quantity <= availableCapacity and
self.batchCapacityResource.batchCapacityInterval->select(a.earliestStartTime <= startTime and
a.quantity <= availableCapacity)->forAll(self.startTime <= startTime)

context UnitCapacityResource inv:
self.hasCapacityConflict IF
self.unitCapacityInterval->exists(availableDuration<0)

context BatchCapacityResource inv:
self.hasCapacityConflict IF
self.batchCapacityInterval->exists(availableCapacity<0)

context ProductionActivity::hasTimeConflict (order : ProductionOrder) : Boolean
post: result = (self.getEarliestStartTime(order) > self.startTime)

context ProductionActivity inv:
self.isScheduled IF
self.productionActivityType->exists() and self.productionActivityType.discreteStateResource->notEmpty() and
self.productionActivityType.discreteStateResource->forAll(capacityInterval->exists
(ci : CapacityInterval | ci.productionActivity->includes(self) and self.capacityInterval->includes(ci))) and self.startTime =
self.capacityInterval->sortedBy(startTime)->first().startTime and
self.endTime = self.capacityInterval->sortedBy(endTime)->last().endTime
def: getDuration (startTime: Date, endTime: Date) : Integer = endTime - startTime
def: getEarliestStartTime (o : ProductionOrder) : Date =
if self.productionActivityType.precedenceInterval[follows]->isEmpty() then
(o.releaseDate) else (self.productionActivityType.precedenceInterval[follows]->first().
productionActivityType.productionActivity->any(productSet.productionOrder = o).endTime +
self.productionActivityType.precedenceInterval[follows]->first().lowerBound)
def: isNextActivity (o : ProductionOrder) : Boolean =
if self.isUnscheduled and (self.productionActivityType.precedenceInterval[follows]->isEmpty() or
self.productionActivityType.precedenceInterval[follows]->first().
productionActivityType.productionActivity->any(productSet.productionOrder = o).isUnscheduled then
(true) else (false).
```

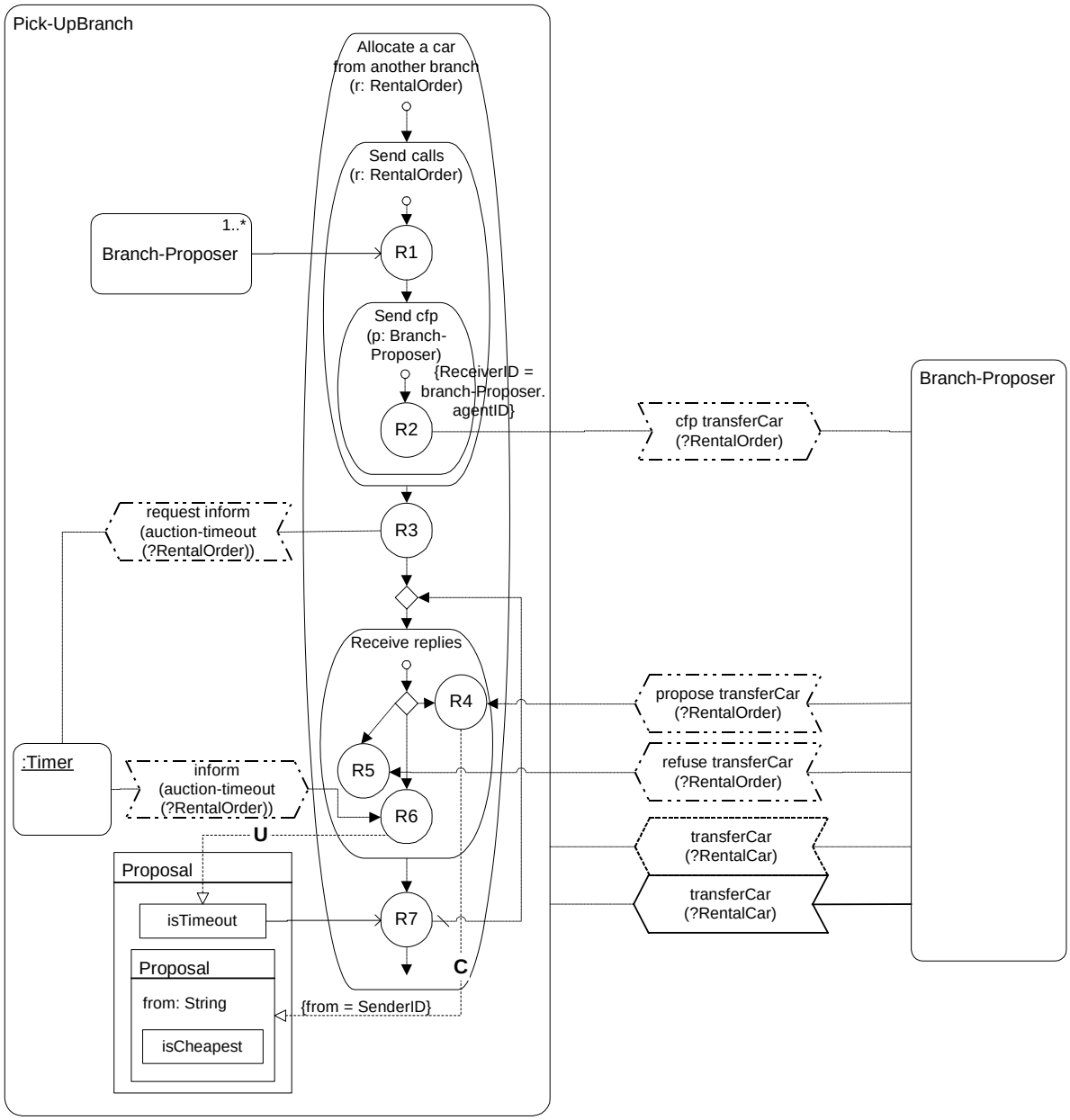
Appendix D (continued). Derivation rules for the case study of the ceramic factory

```
context ProductionOrder inv:
self.isScheduled IF
self.productType->exists() and
self.productSet->exists() and
self.productType.productionActivityType->notEmpty() and
self.productType.productionActivityType->forAll(t : productionActivityType | t.productionActivity->exists
(a: ProductionActivity | a.isScheduled and a.typeName = t.activityName and a.productionActivityType = t and
self.productSet->includes(a) and a.productSet = self.productSet))
self.isCompleted IF
self.productSet.productionActivity->forAll(isCompleted)
```

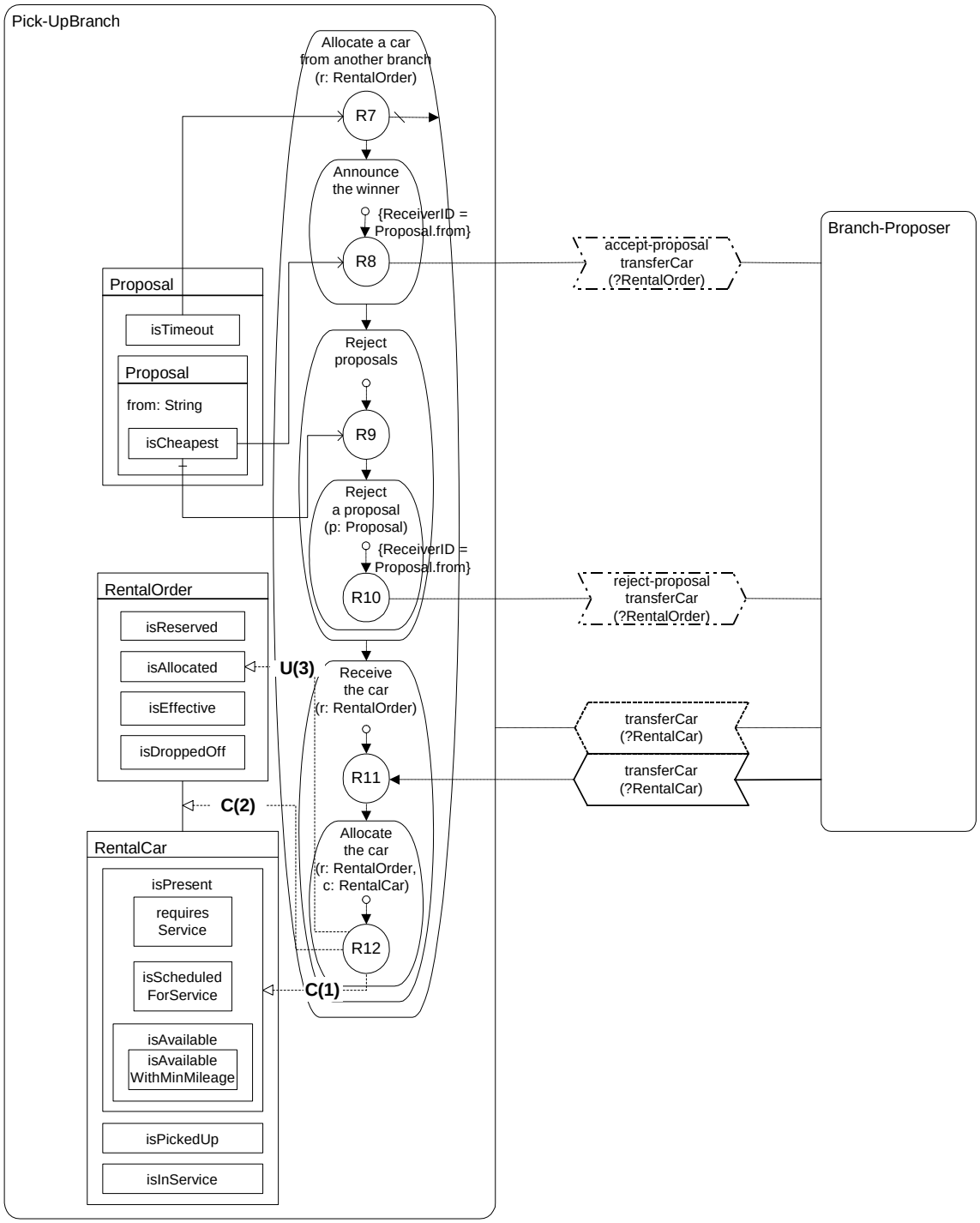
Appendix E. AOR activity diagrams for the case study of car rental



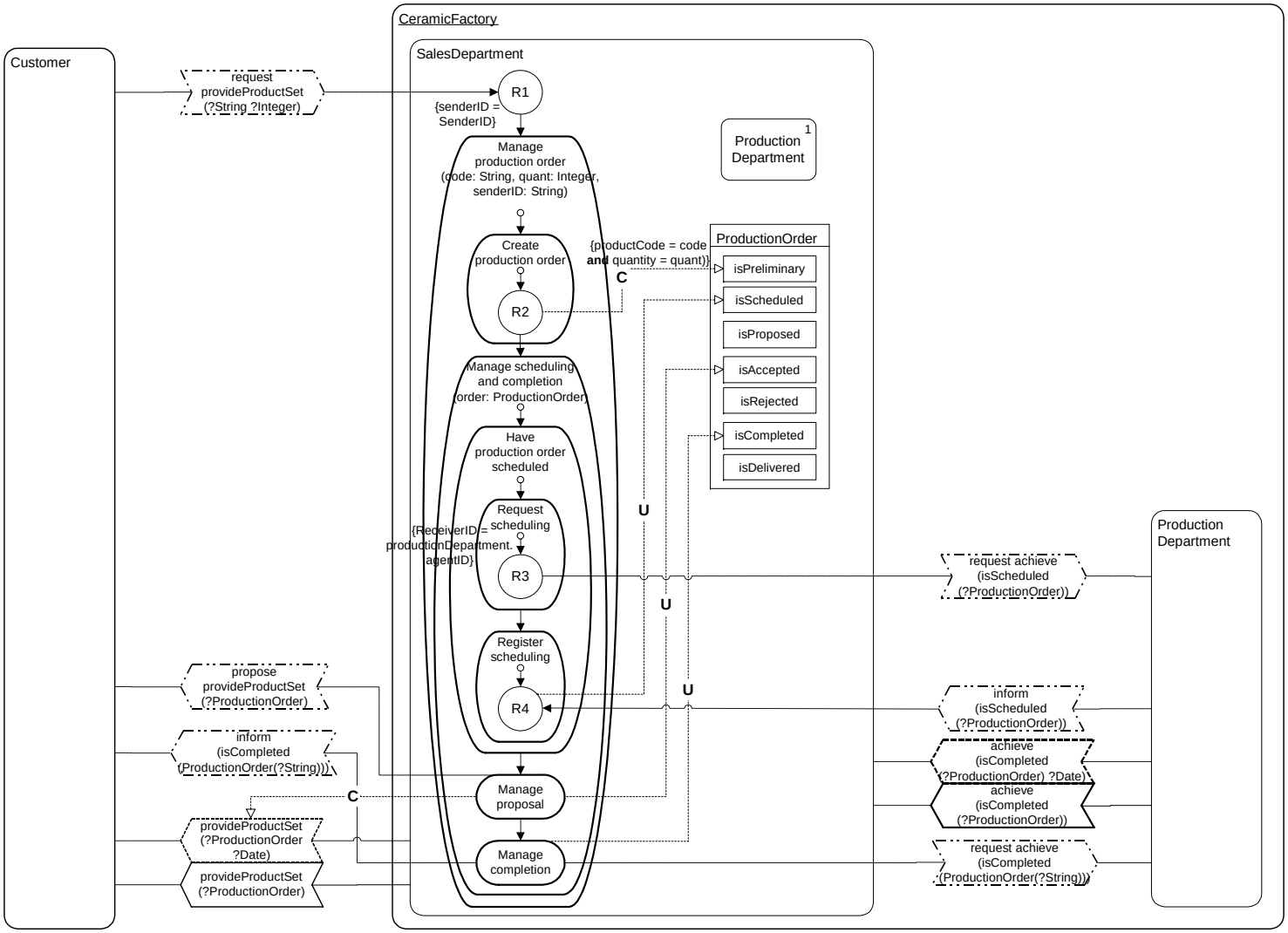
Appendix E (continued). AOR activity diagrams for the case study of car rental



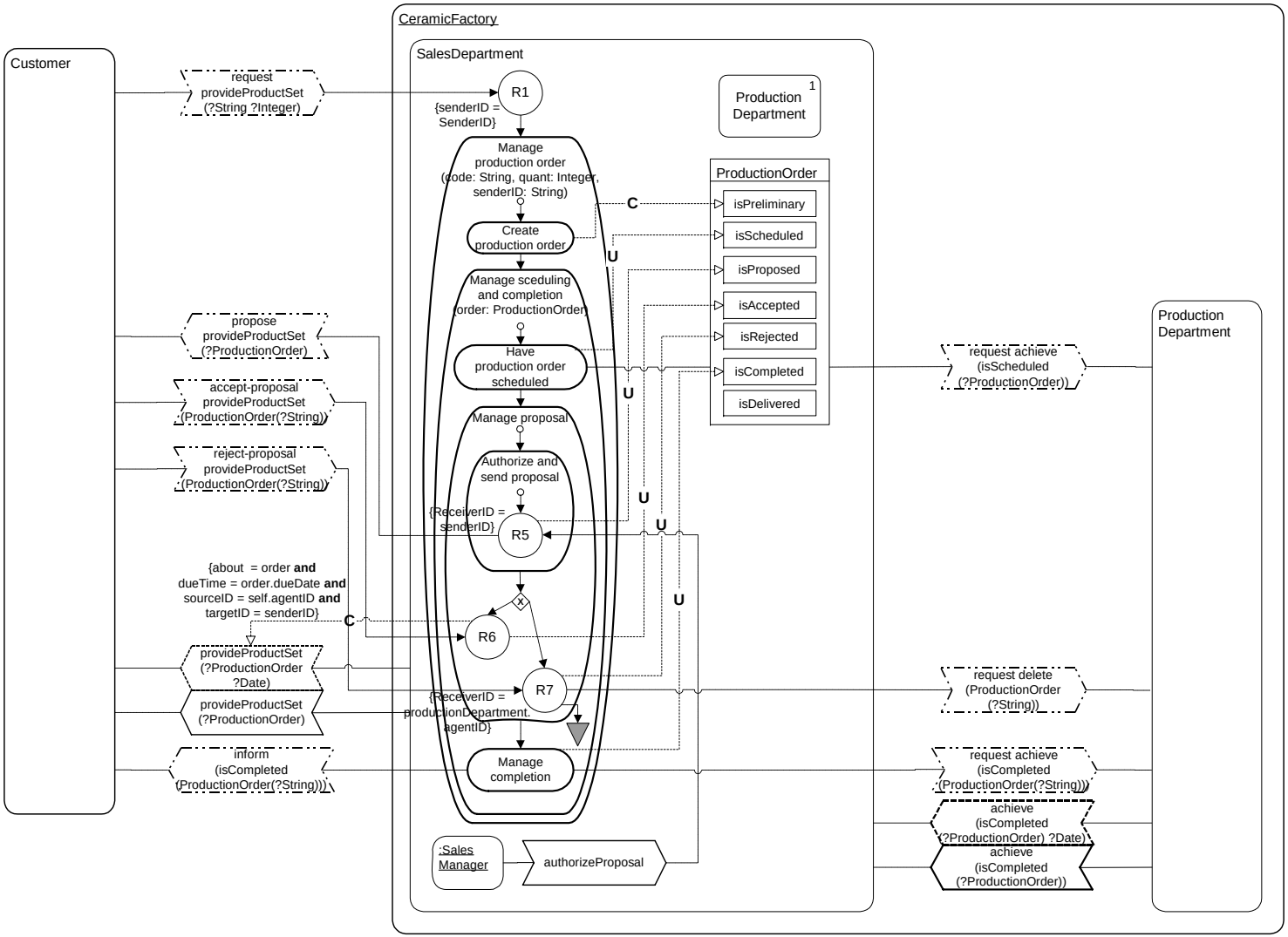
Appendix E (continued). AOR activity diagrams for the case study of car rental



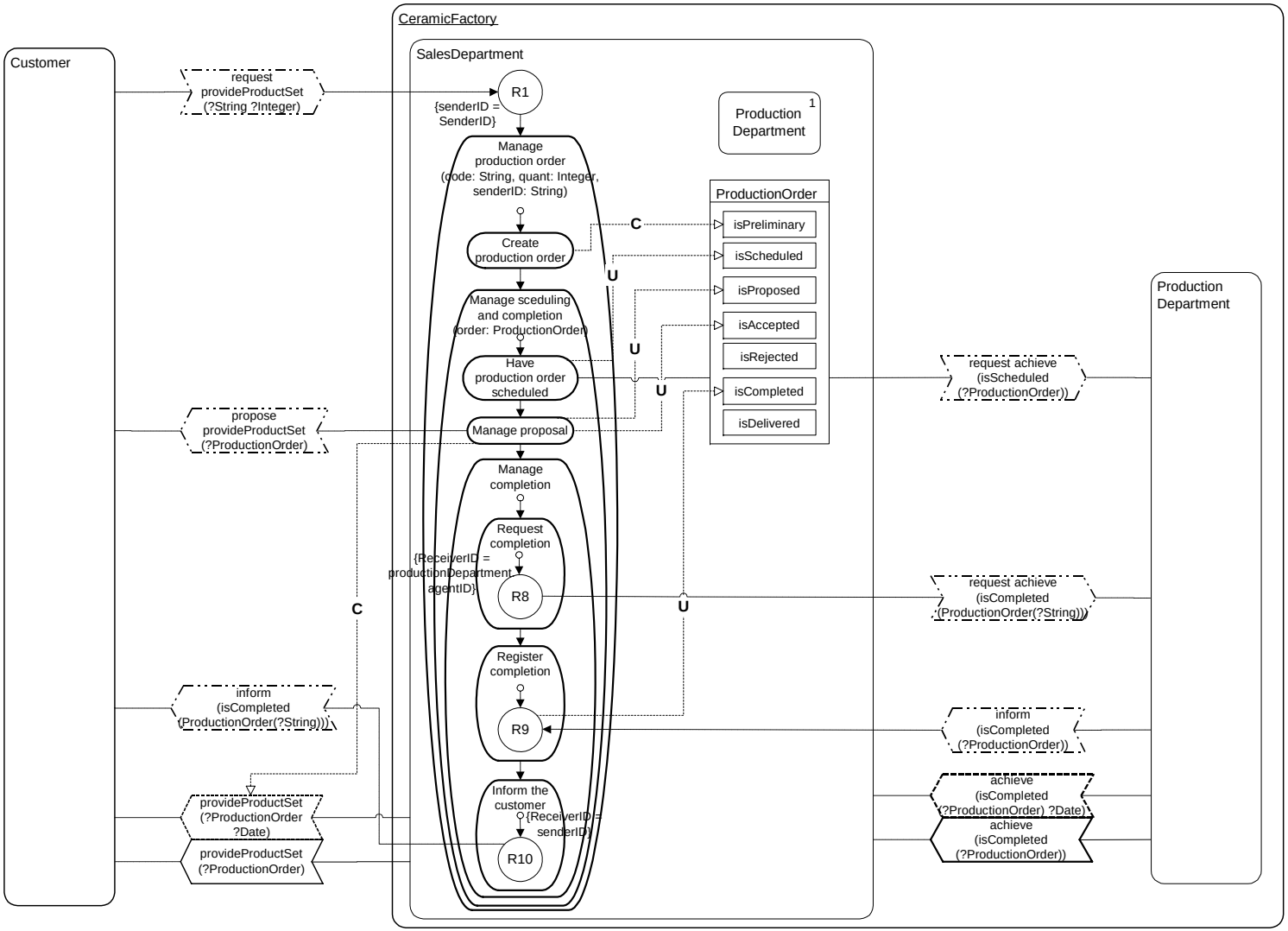
Appendix F. AOR activity diagrams for the case study of the ceramic factory



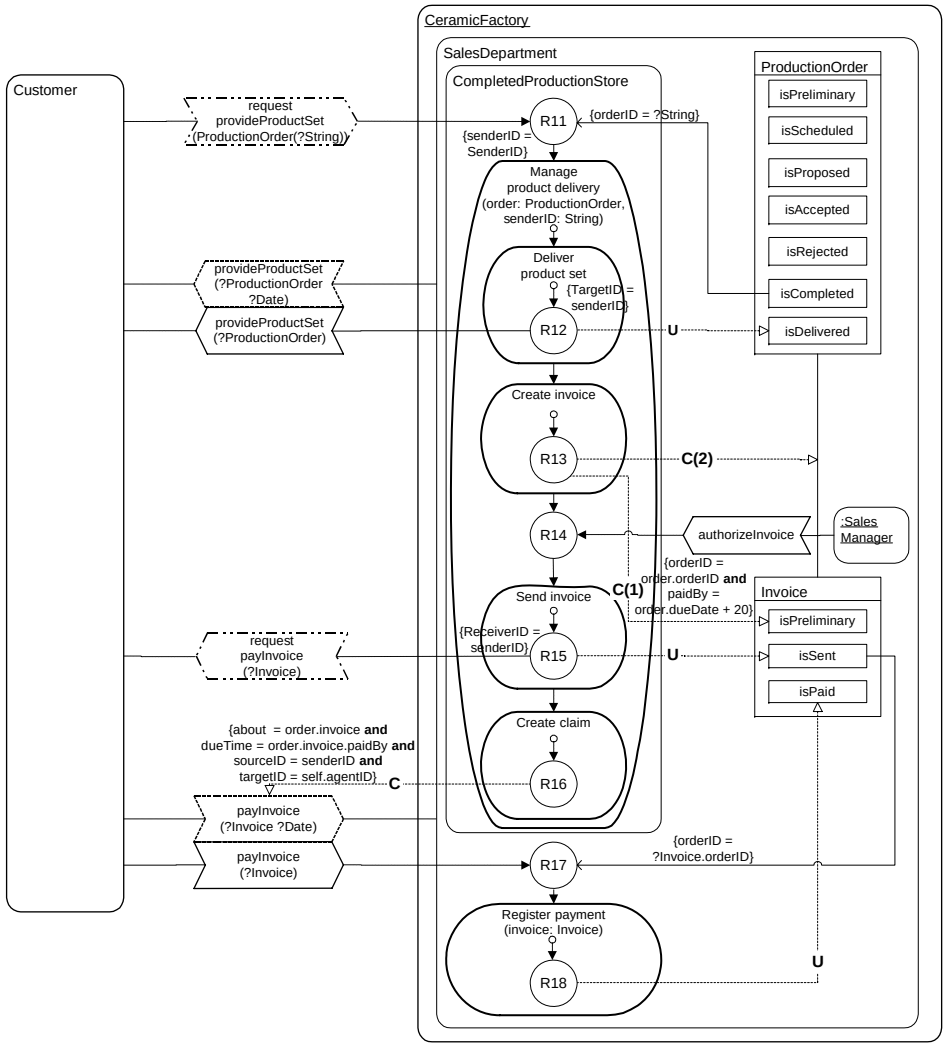
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



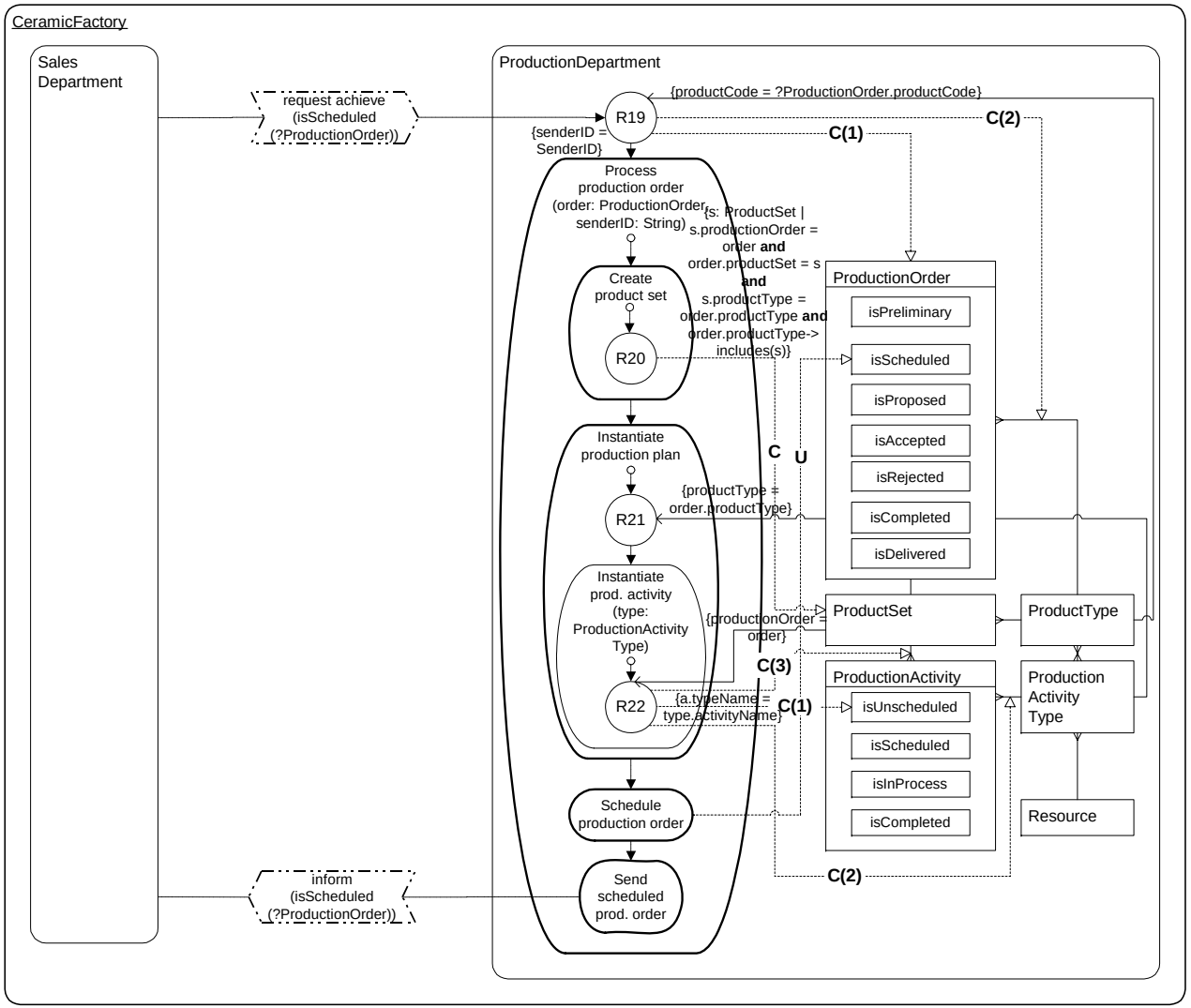
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



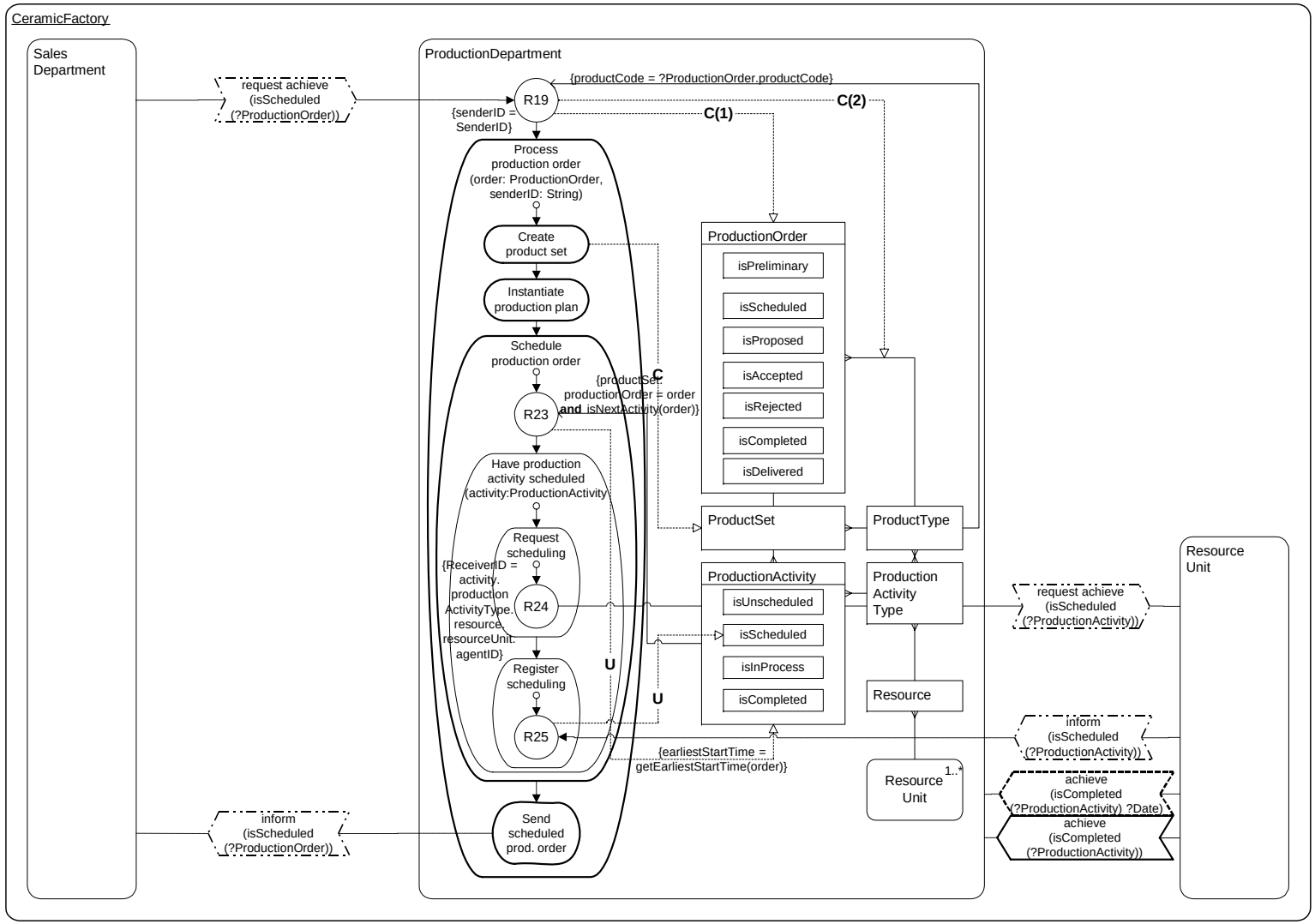
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



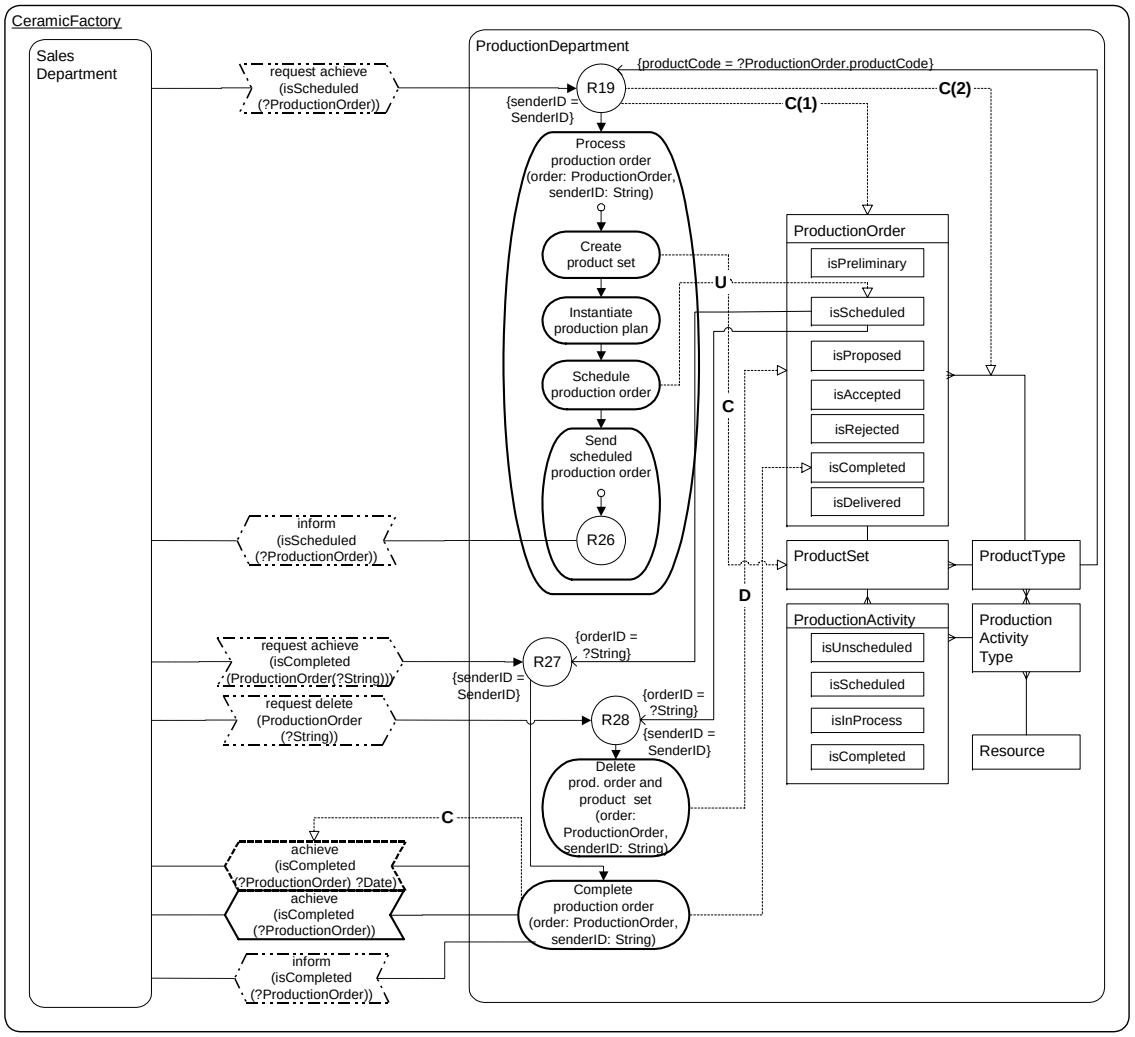
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



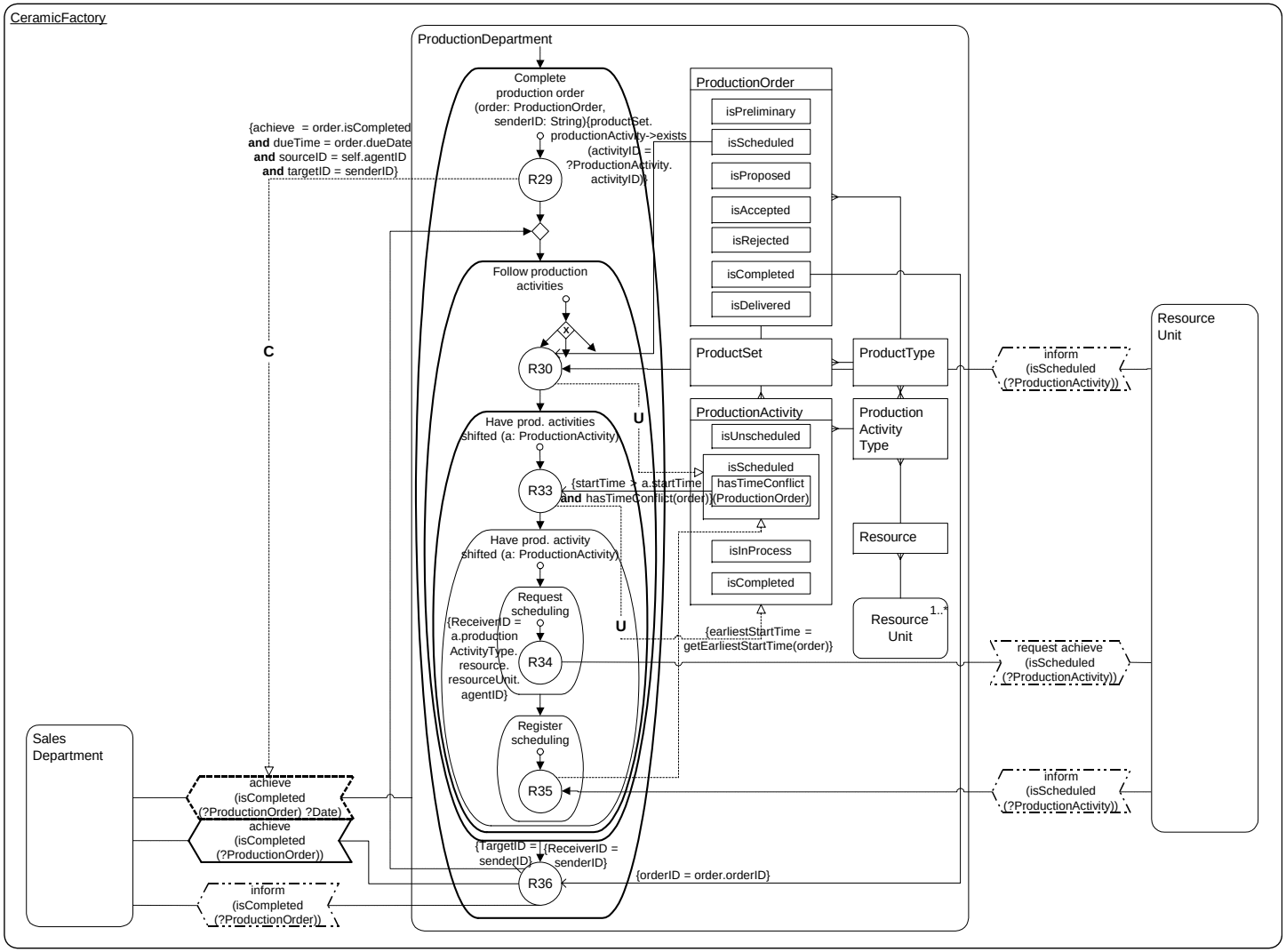
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



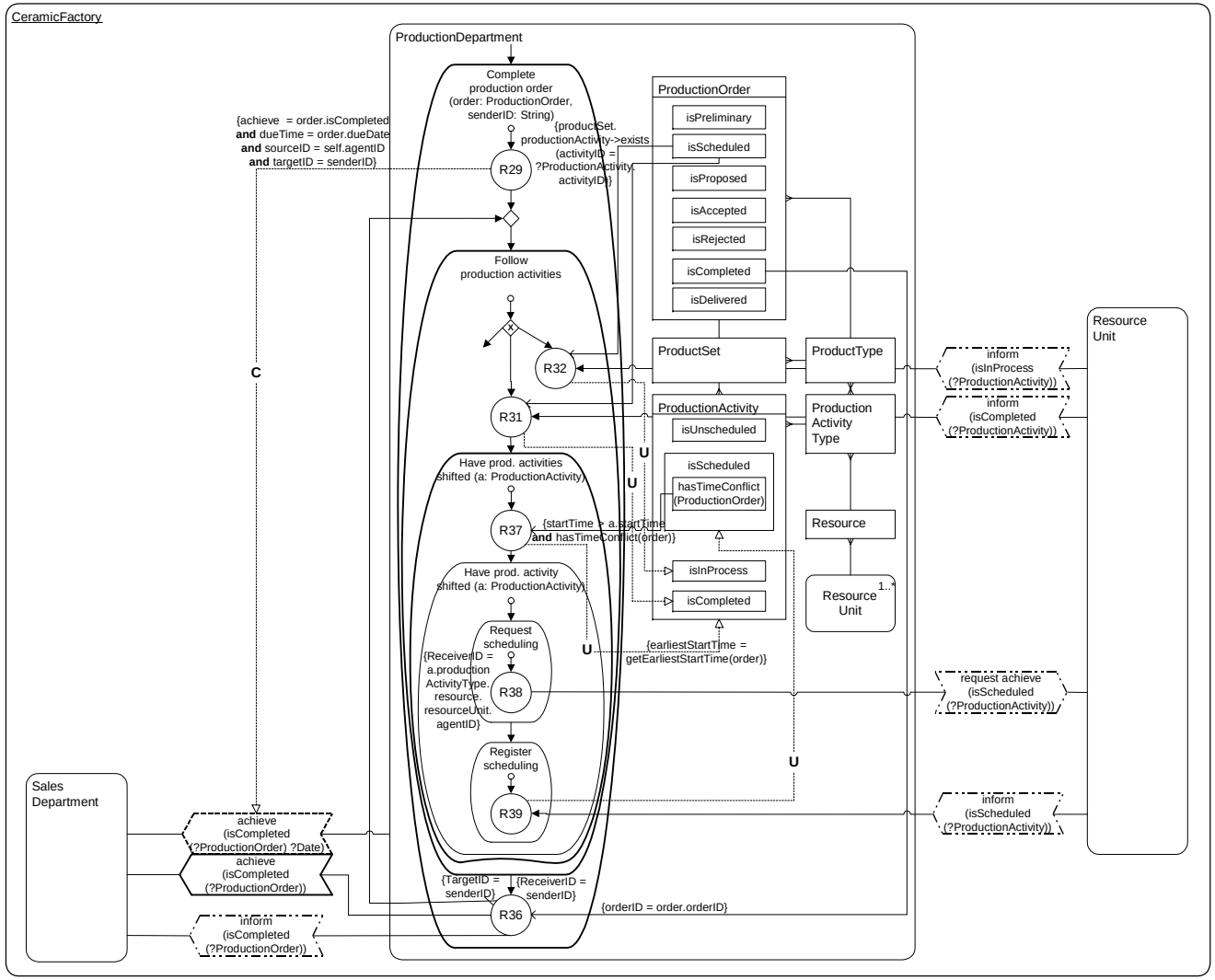
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



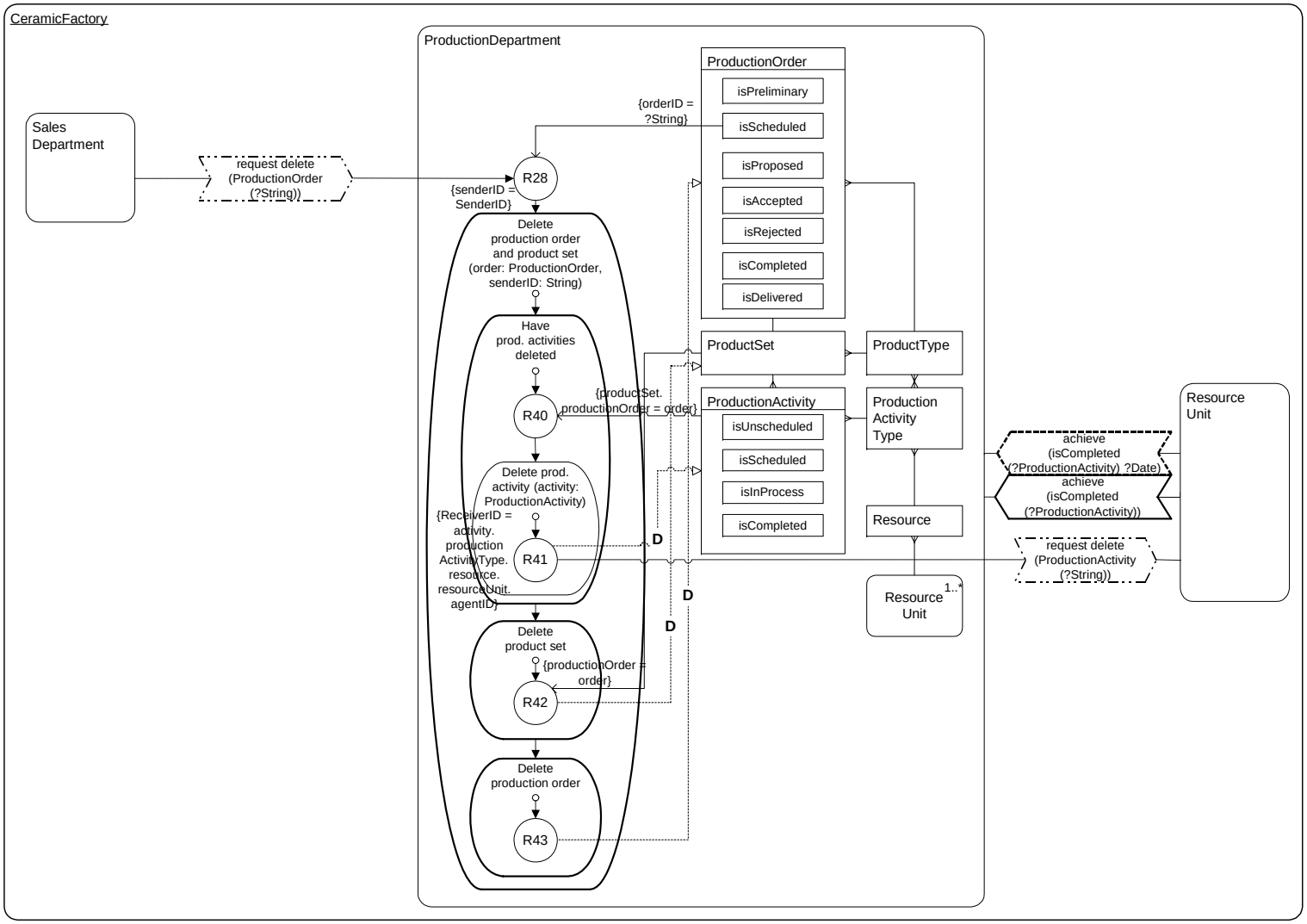
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



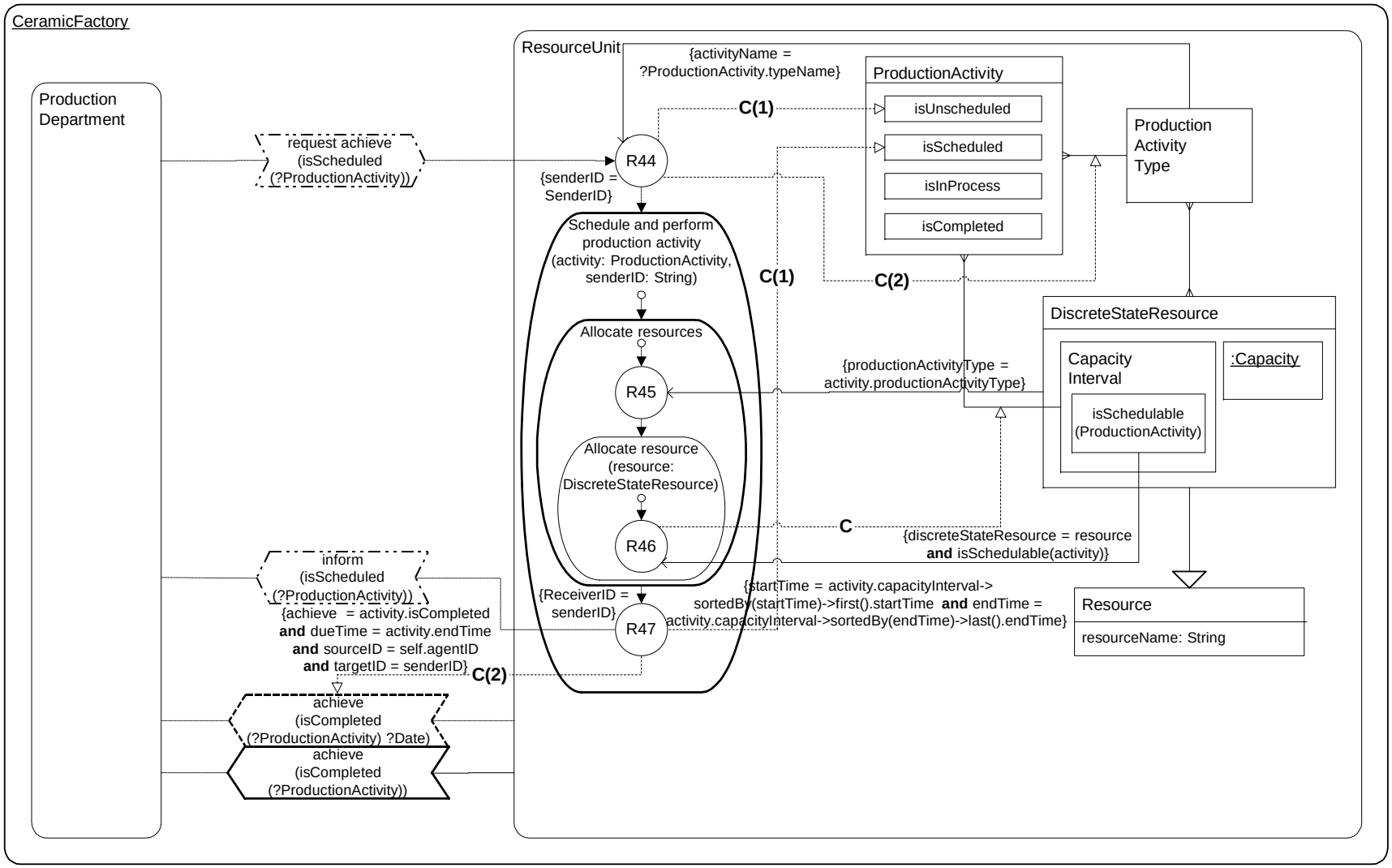
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



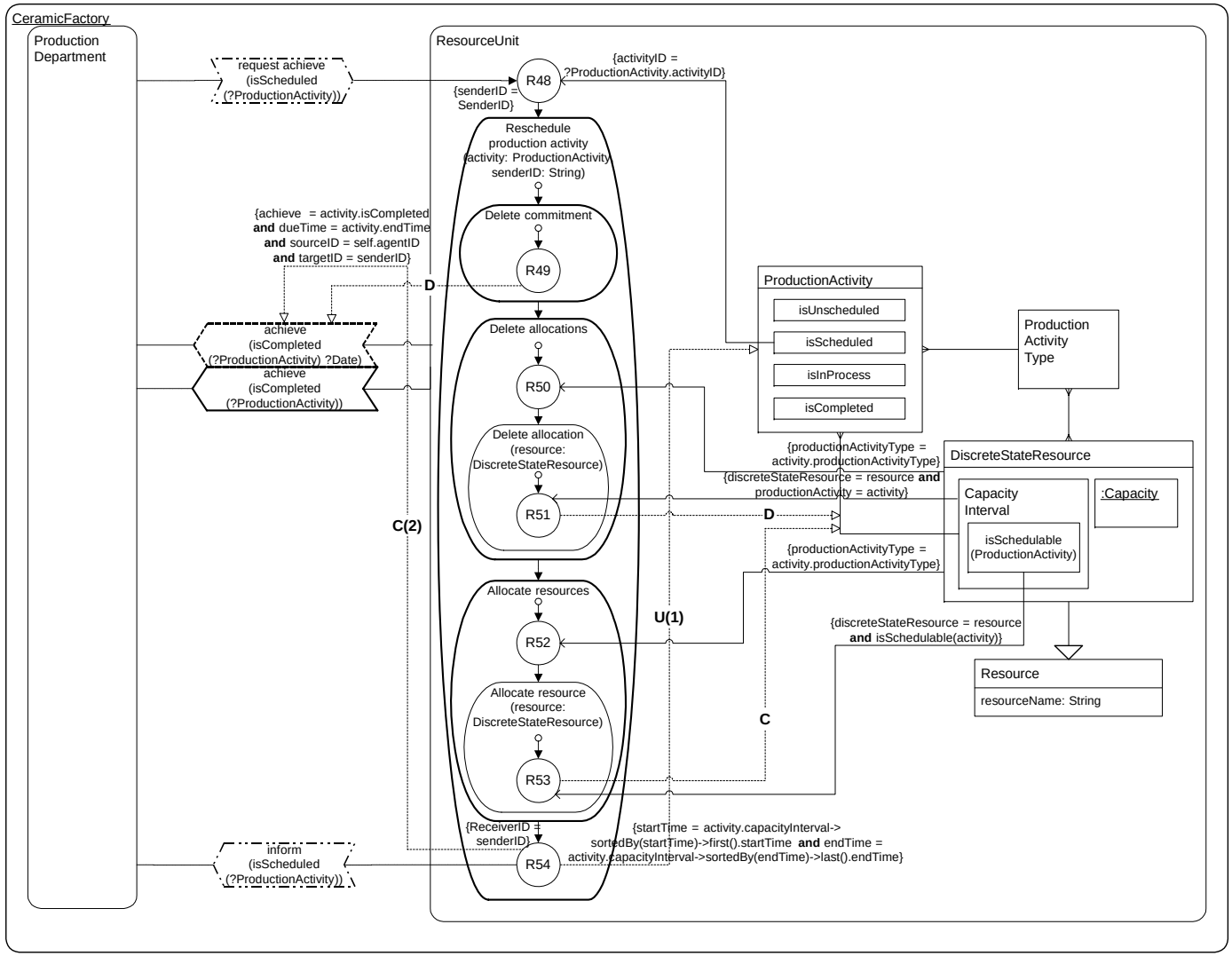
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



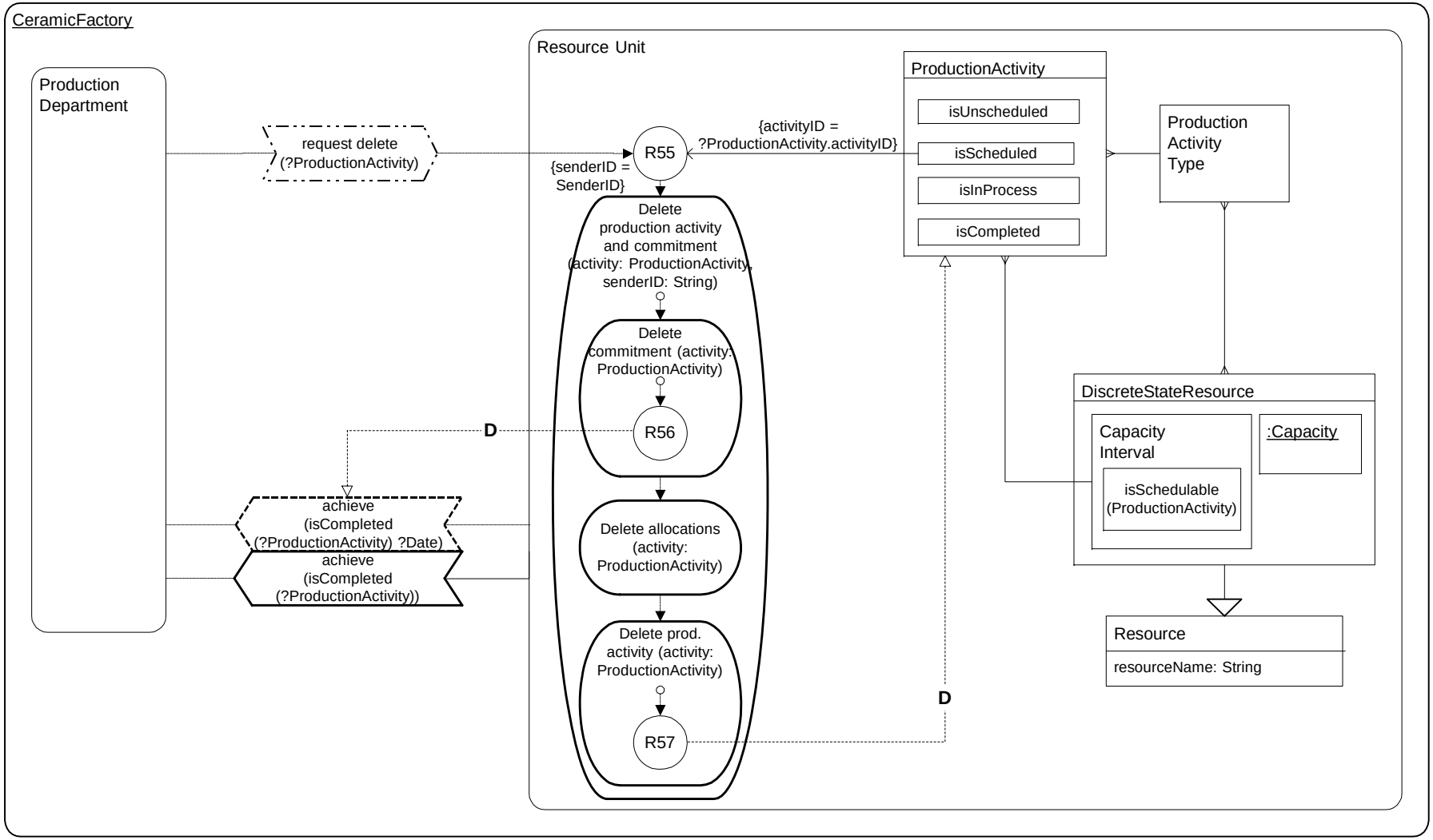
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



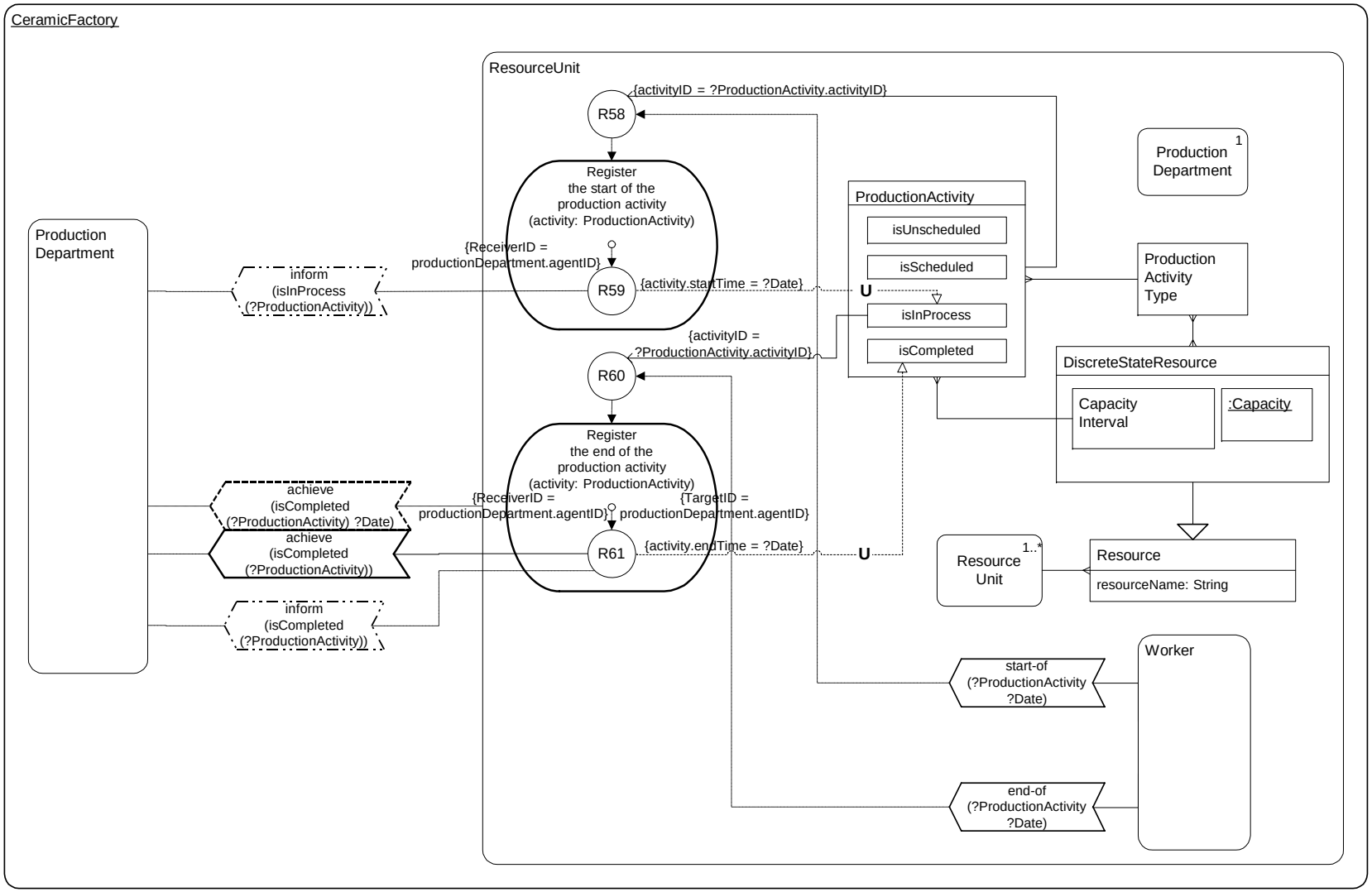
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



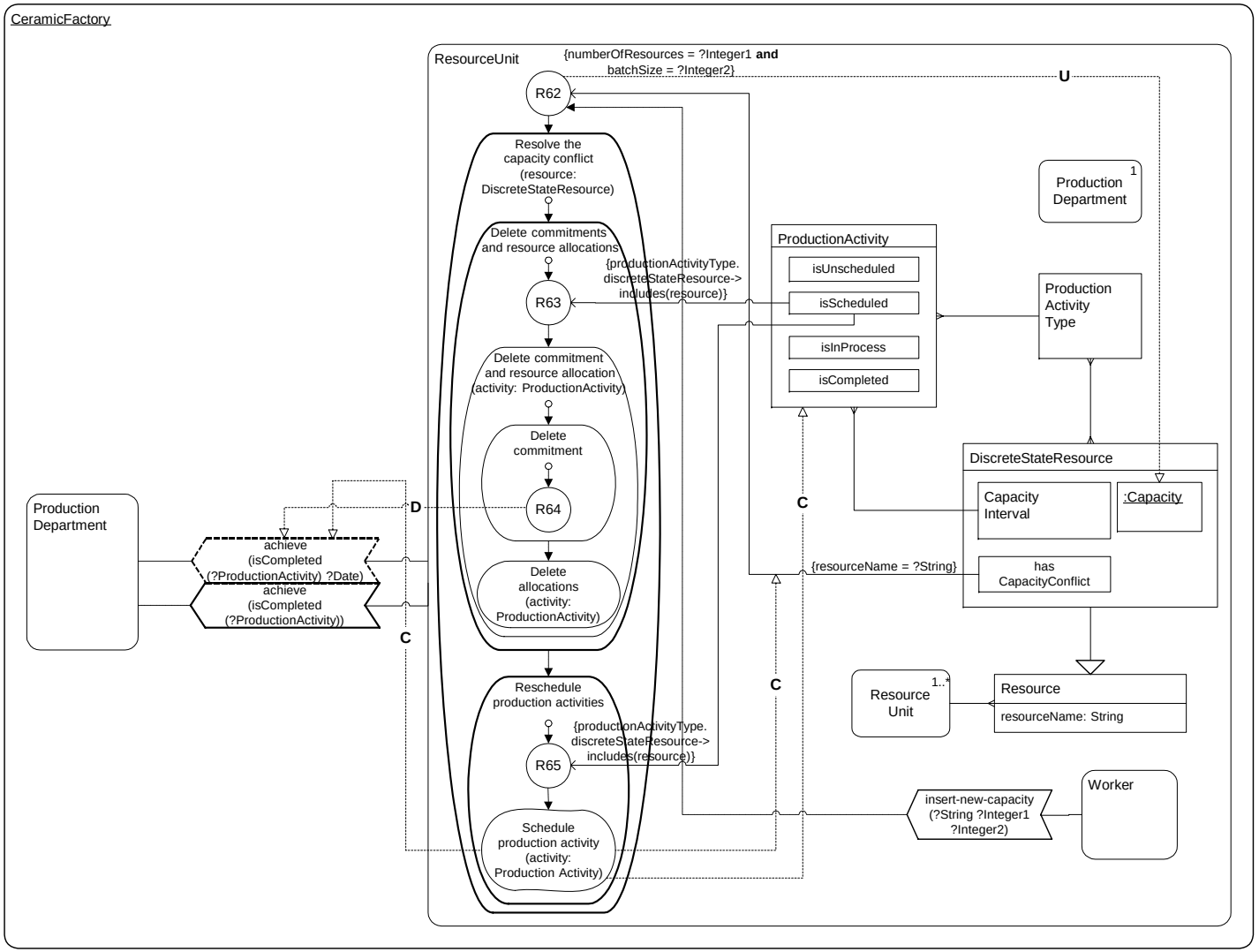
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



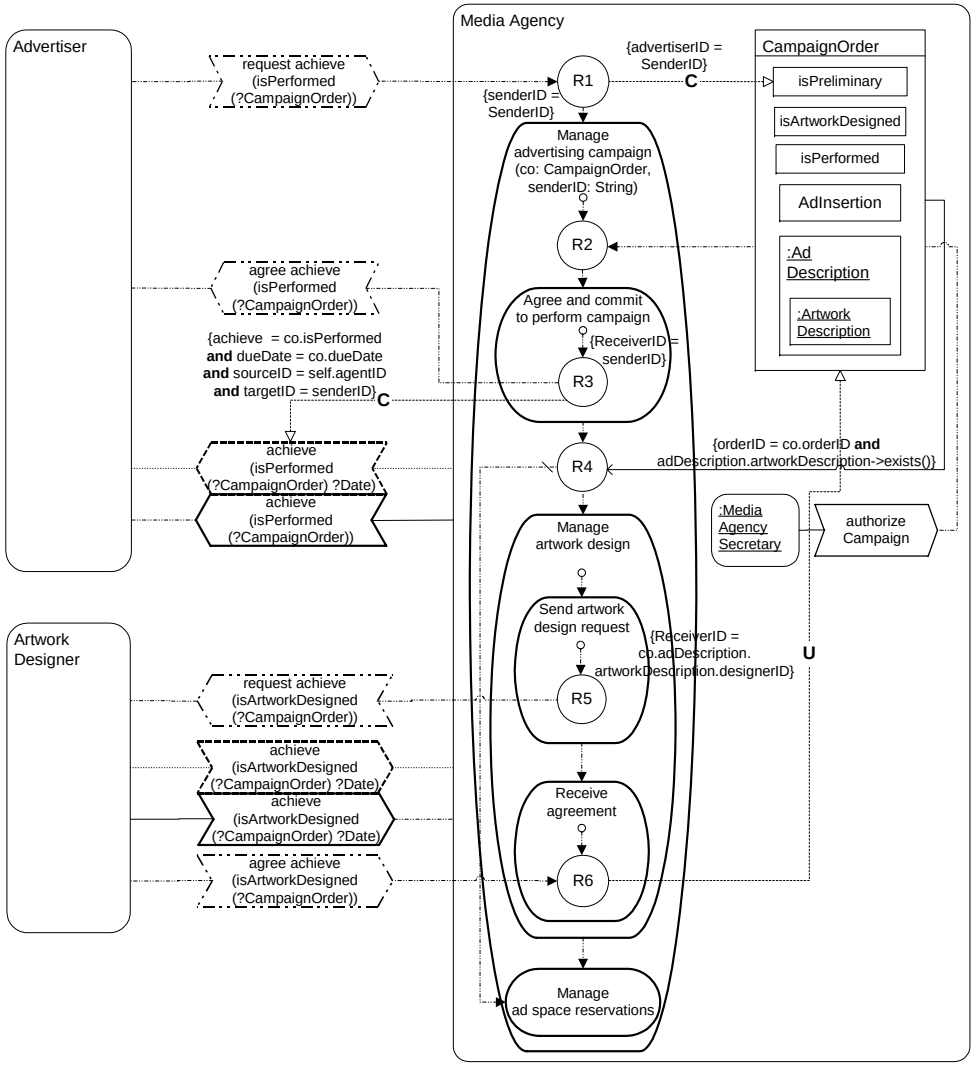
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



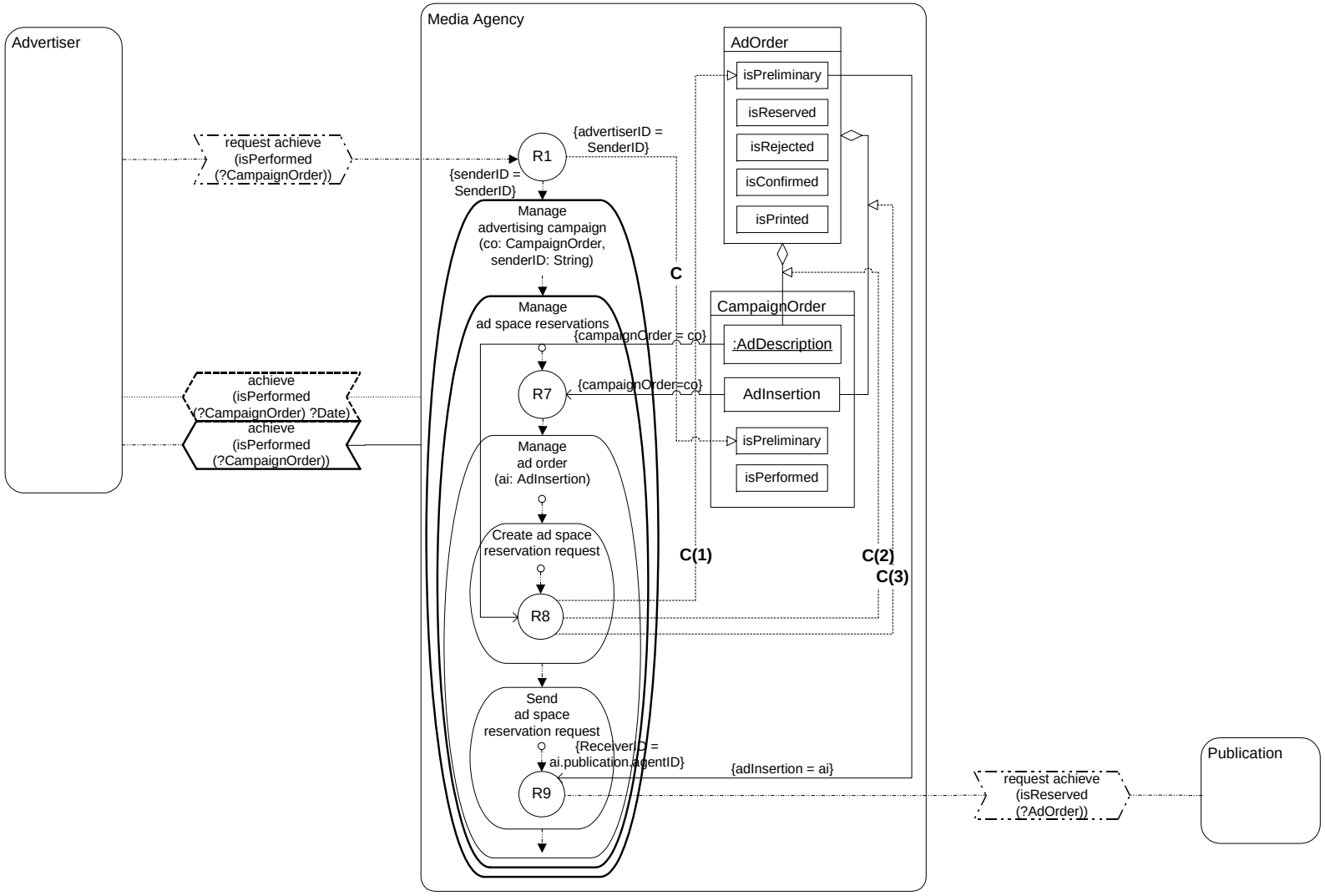
Appendix F (continued). AOR activity diagrams for the case study of the ceramic factory



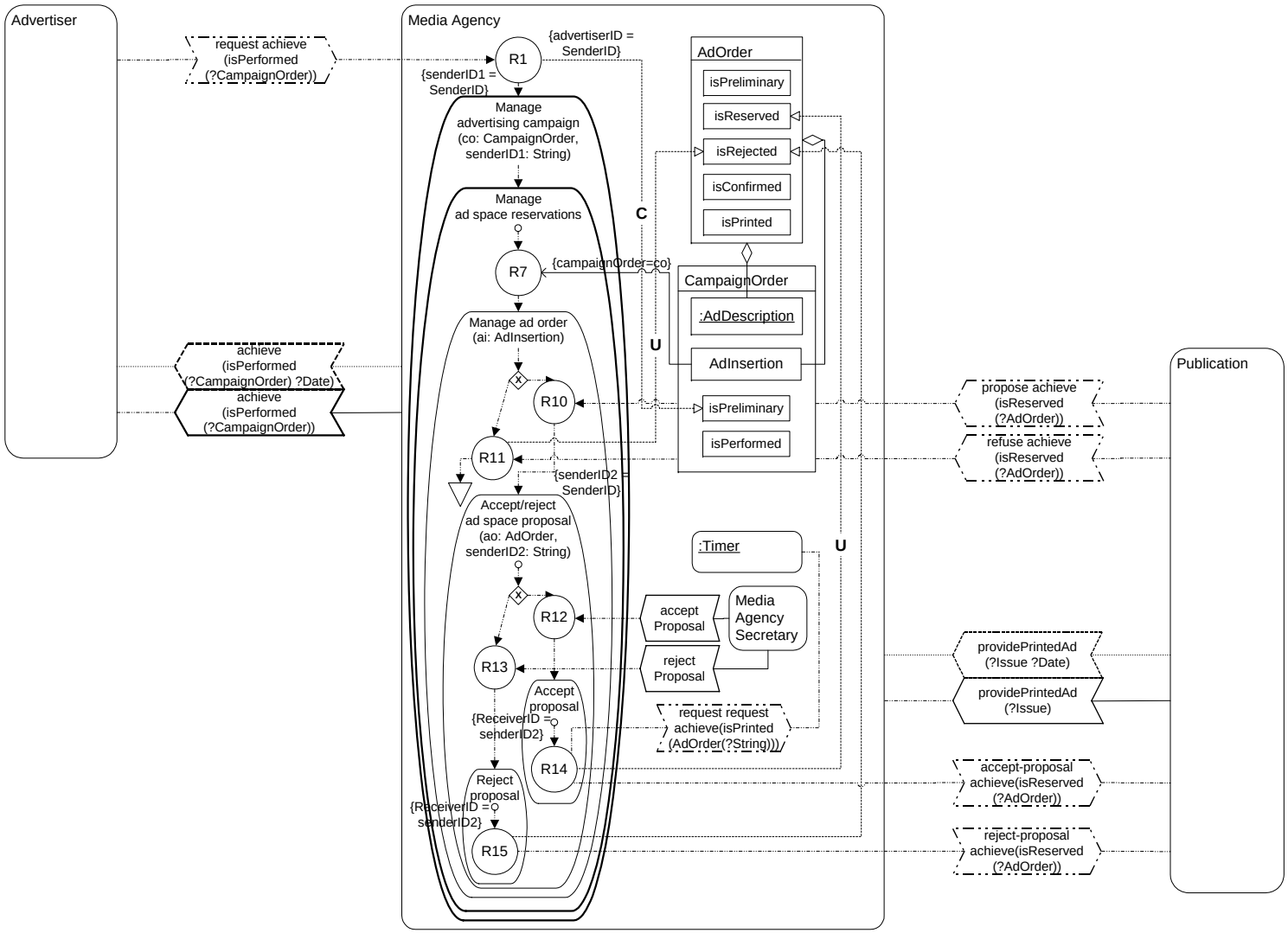
Appendix G. AOR activity diagrams for the case study of advertising



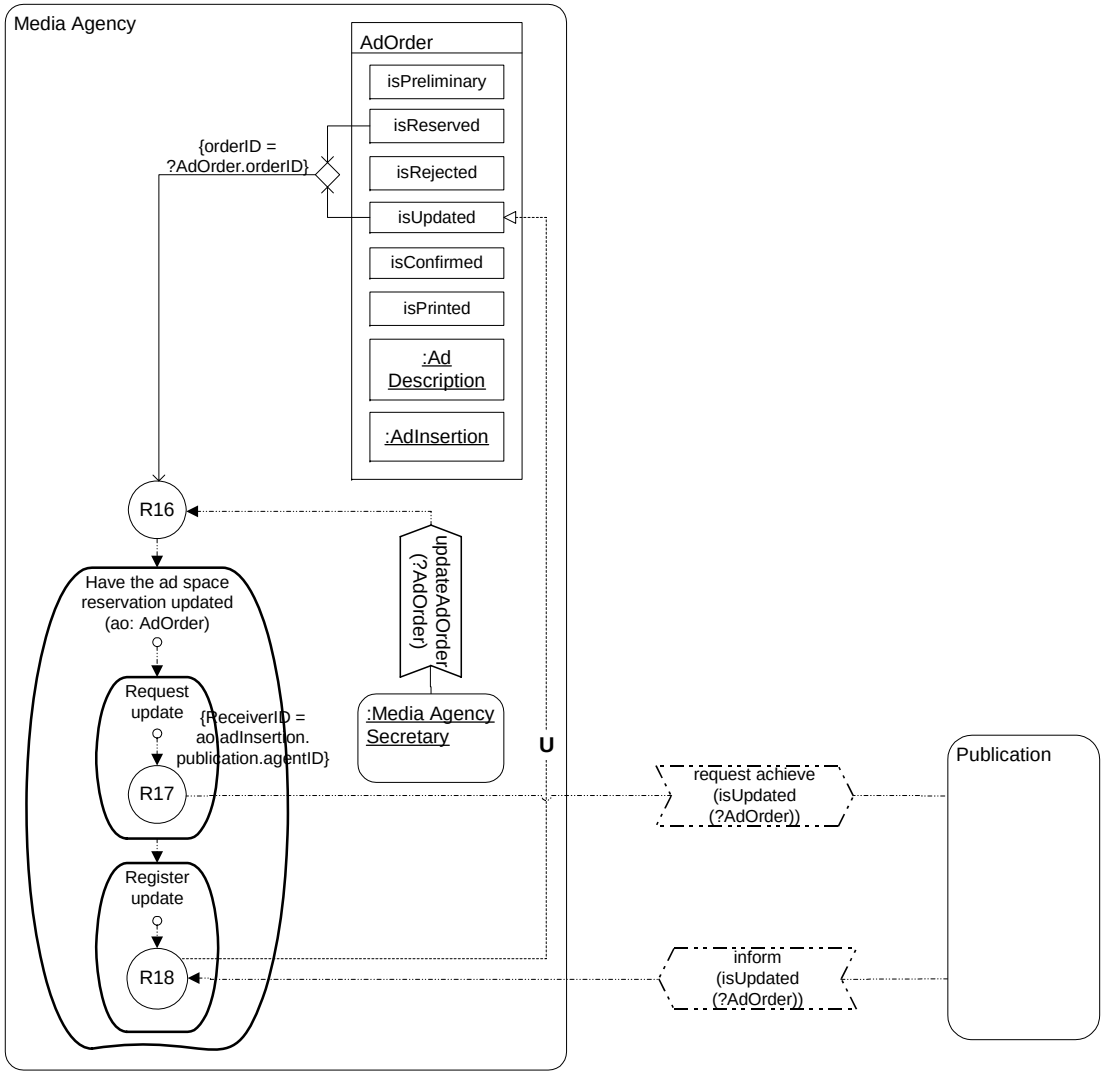
Appendix G (continued). AOR activity diagrams for the case study of advertising



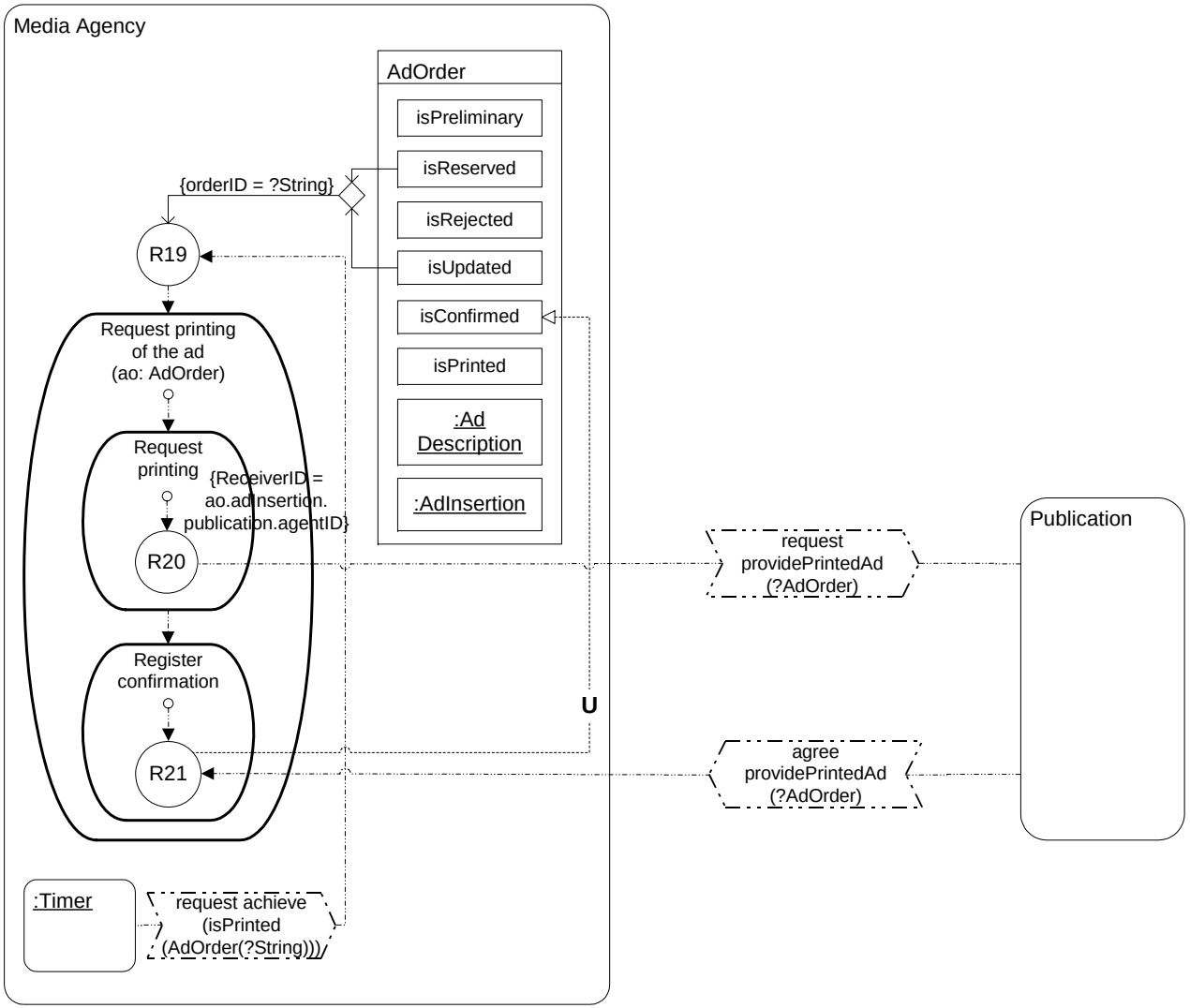
Appendix G (continued). AOR activity diagrams for the case study of advertising



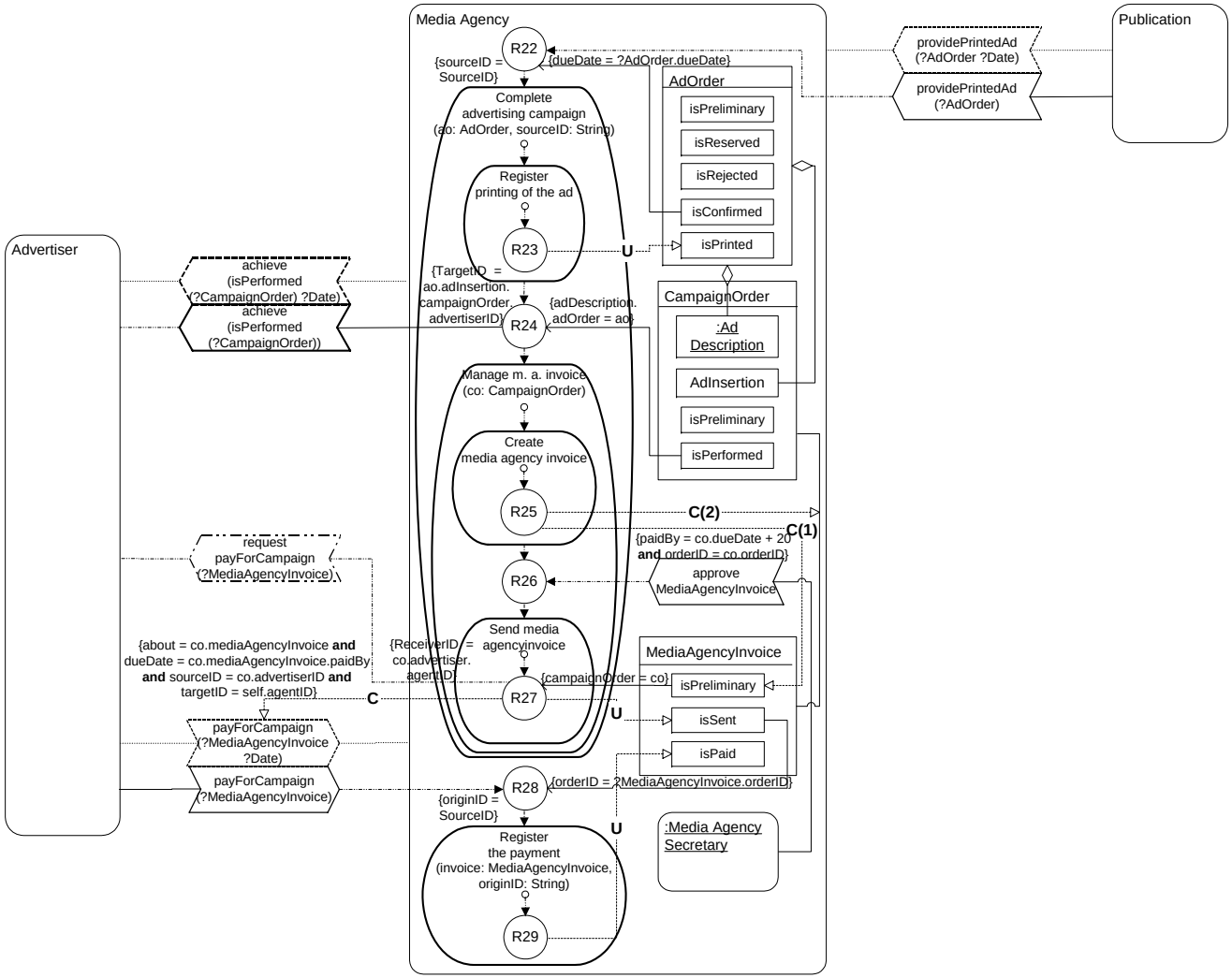
Appendix G (continued). AOR activity diagrams for the case study of advertising



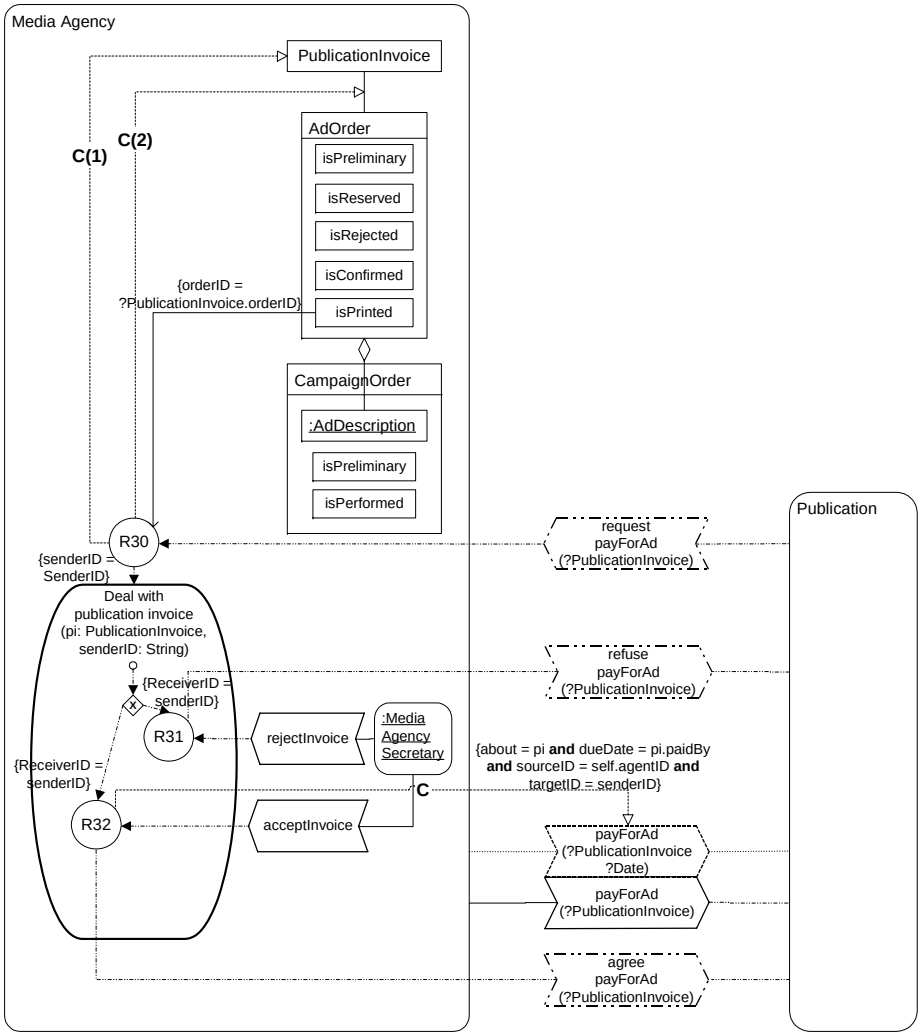
Appendix G (continued). AOR activity diagrams for the case study of advertising



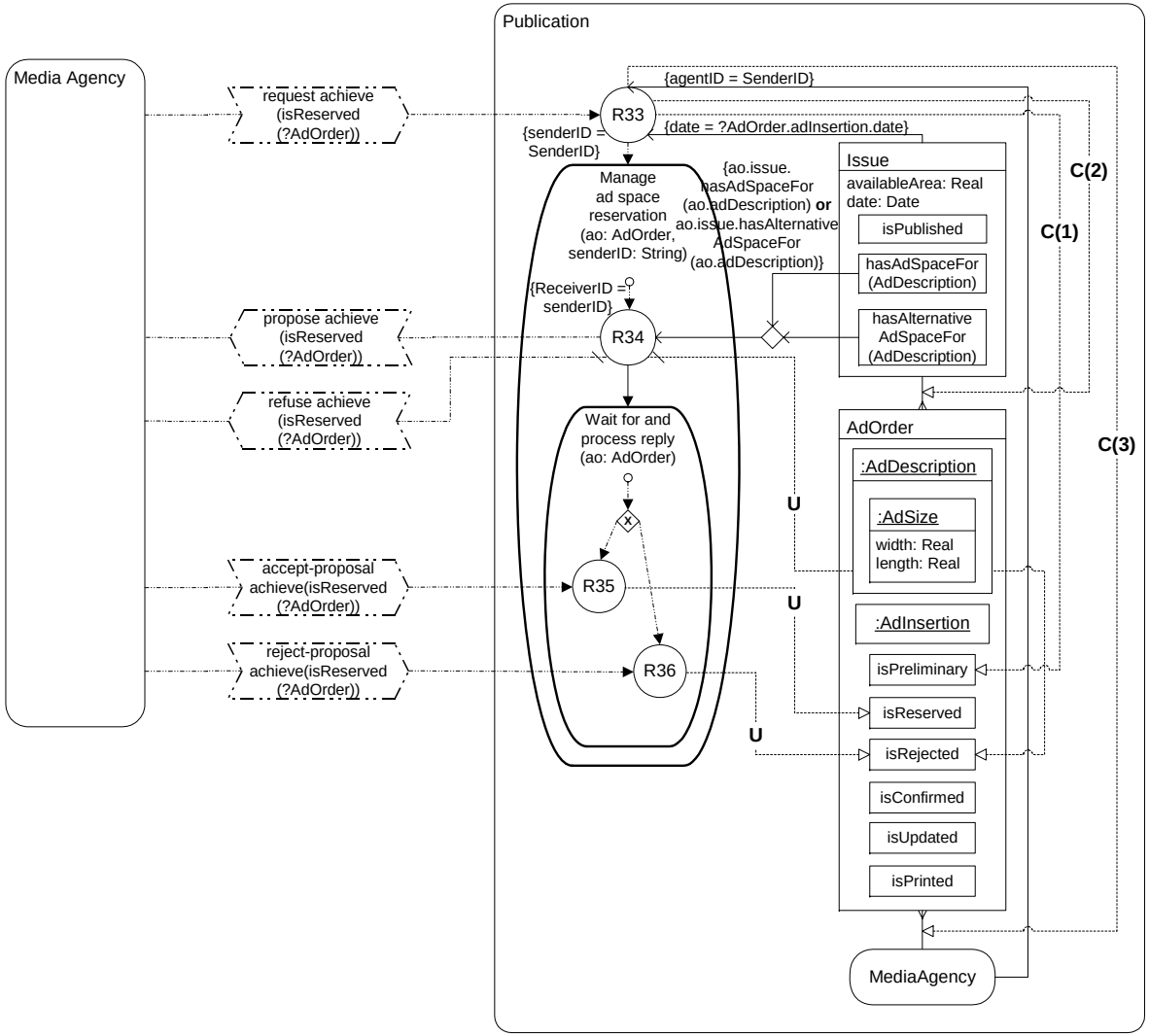
Appendix G (continued). AOR activity diagrams for the case study of advertising



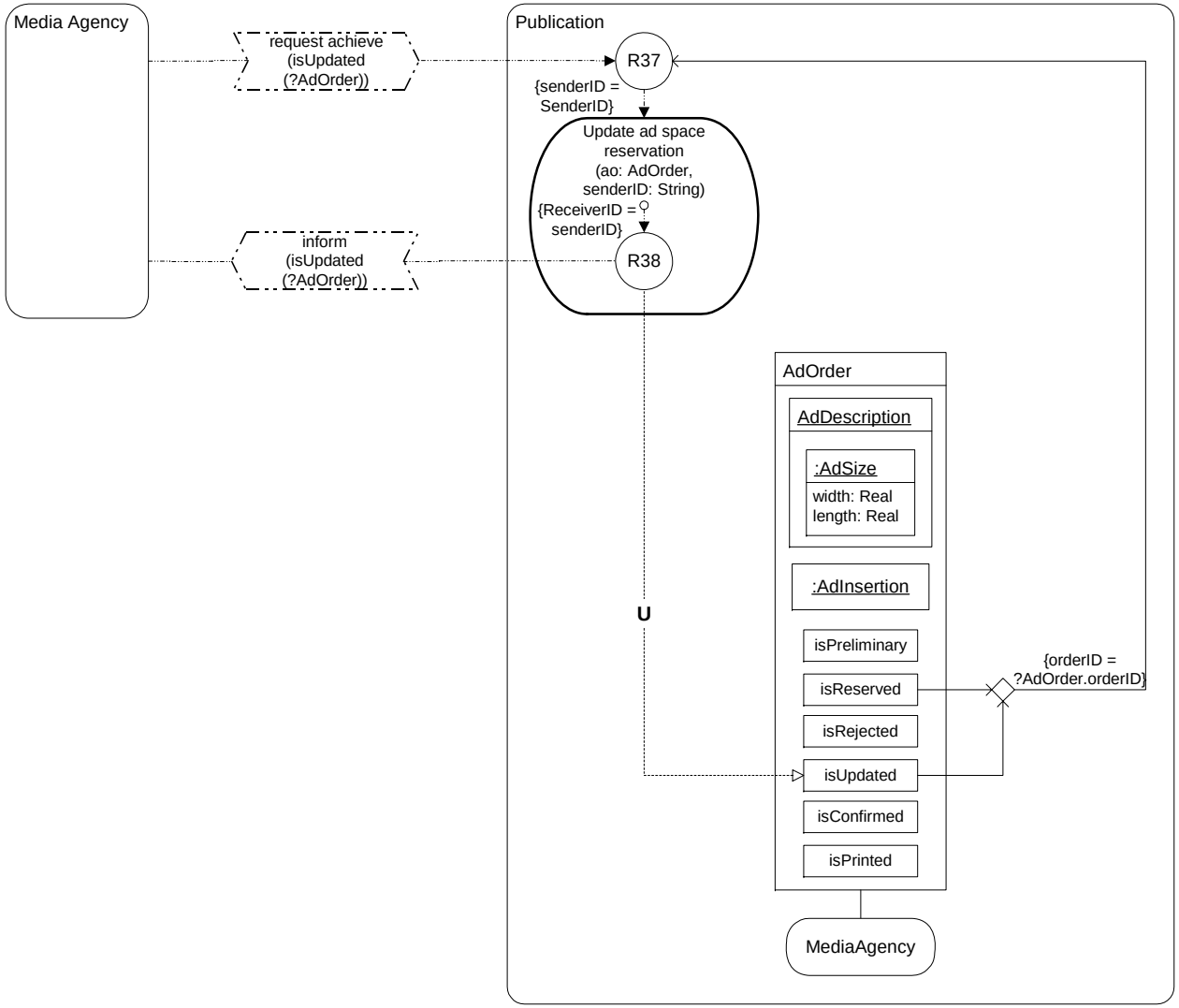
Appendix G (continued). AOR activity diagrams for the case study of advertising



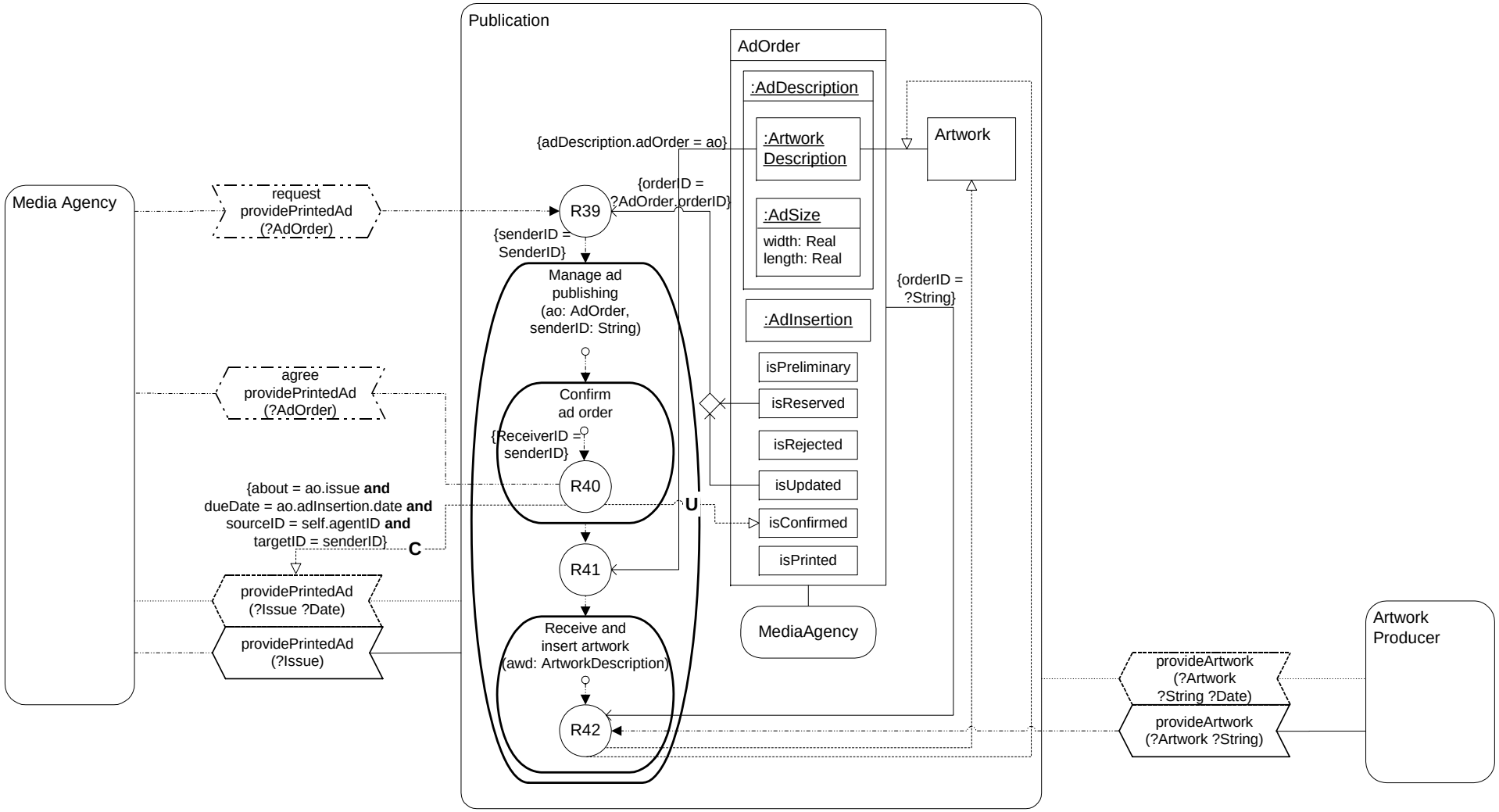
Appendix G (continued). AOR activity diagrams for the case study of advertising



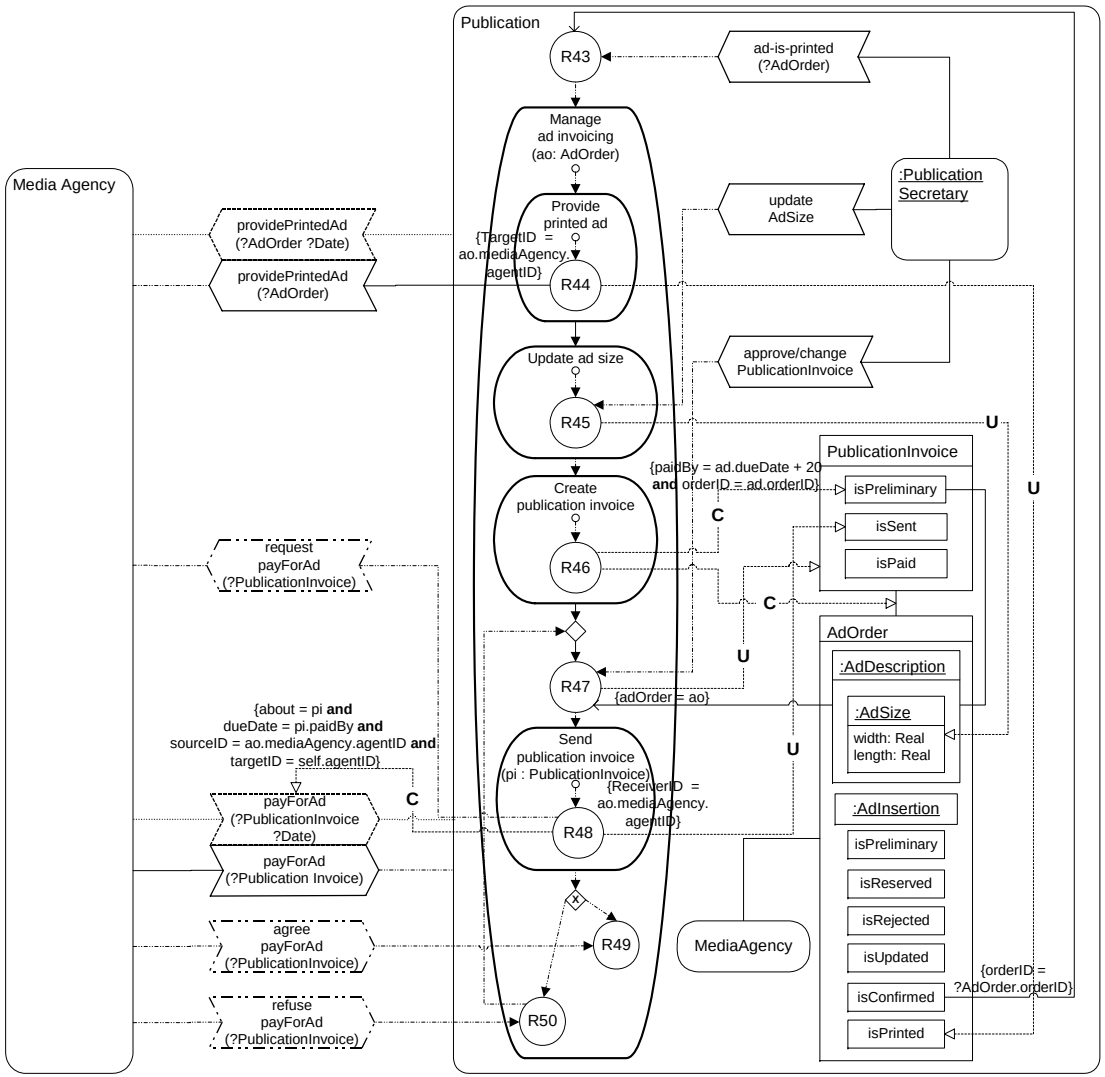
Appendix G (continued). AOR activity diagrams for the case study of advertising



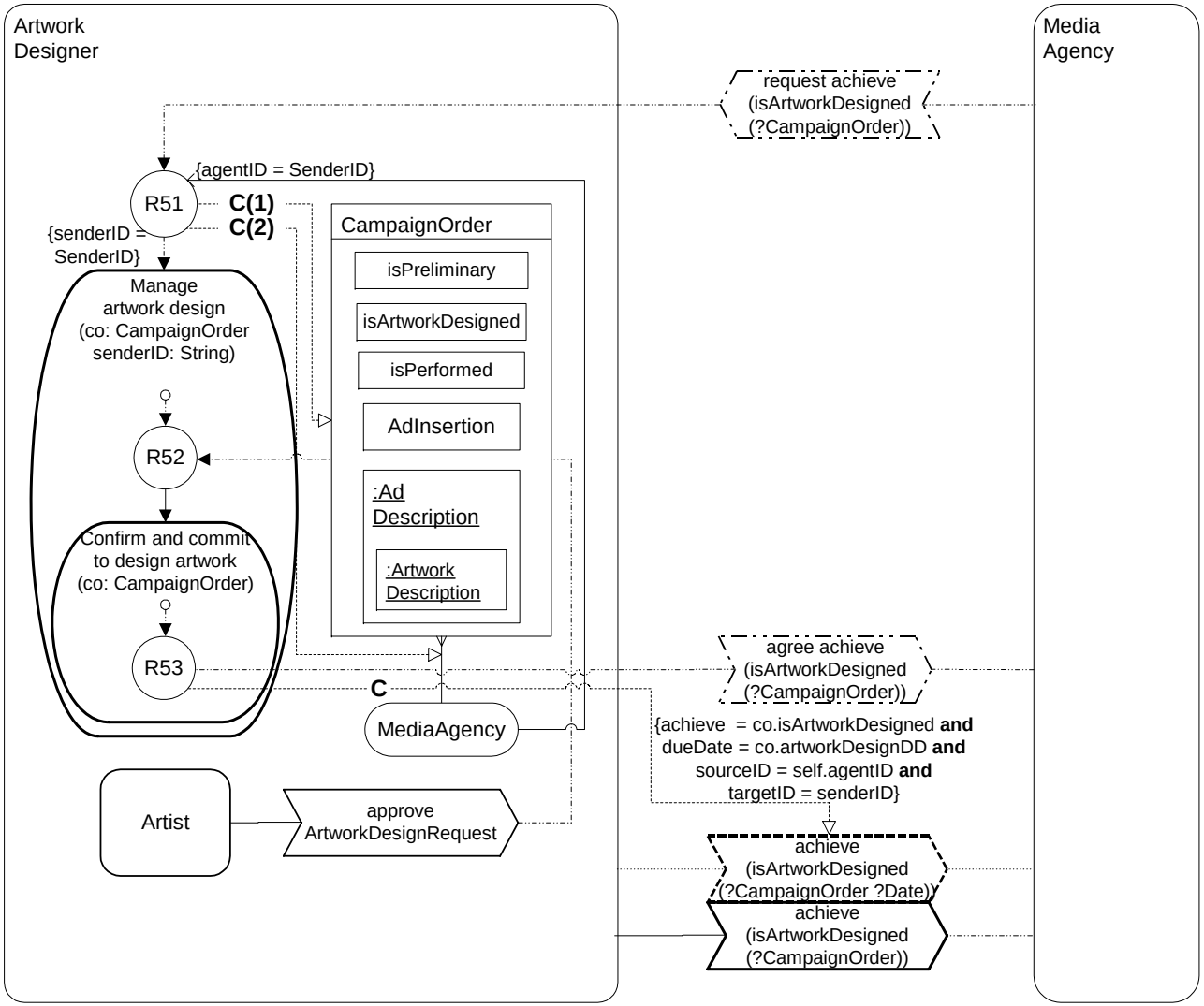
Appendix G (continued). AOR activity diagrams for the case study of advertising



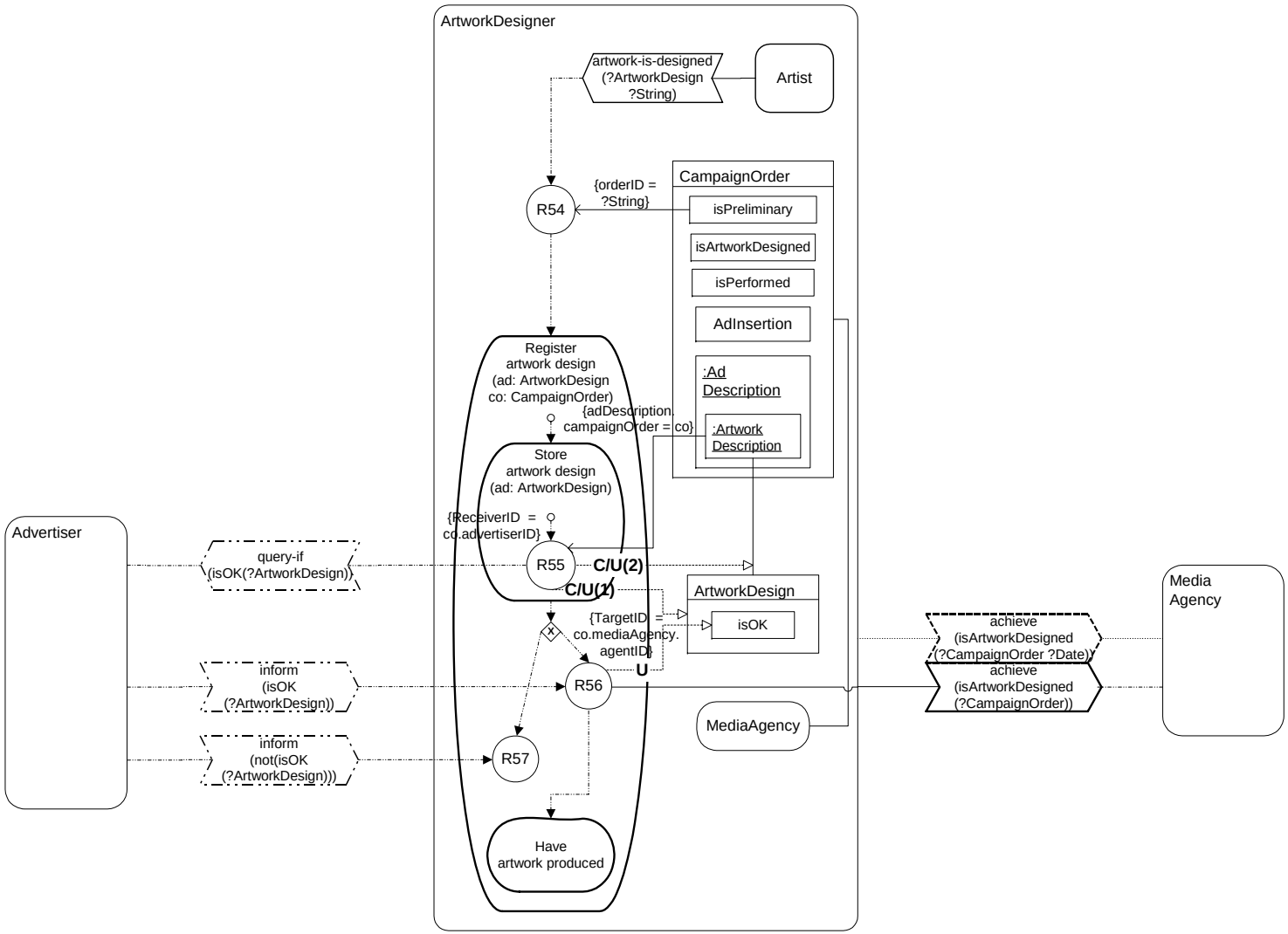
Appendix G (continued). AOR activity diagrams for the case study of advertising



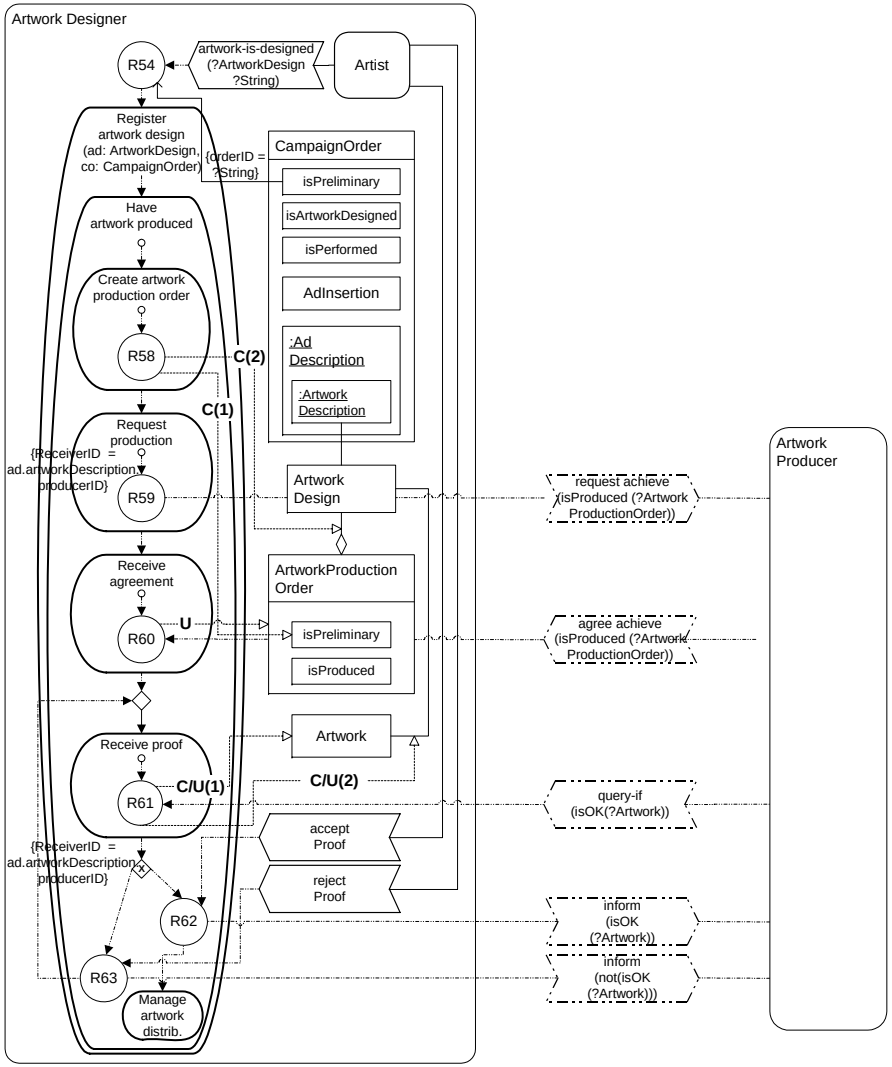
Appendix G (continued). AOR activity diagrams for the case study of advertising



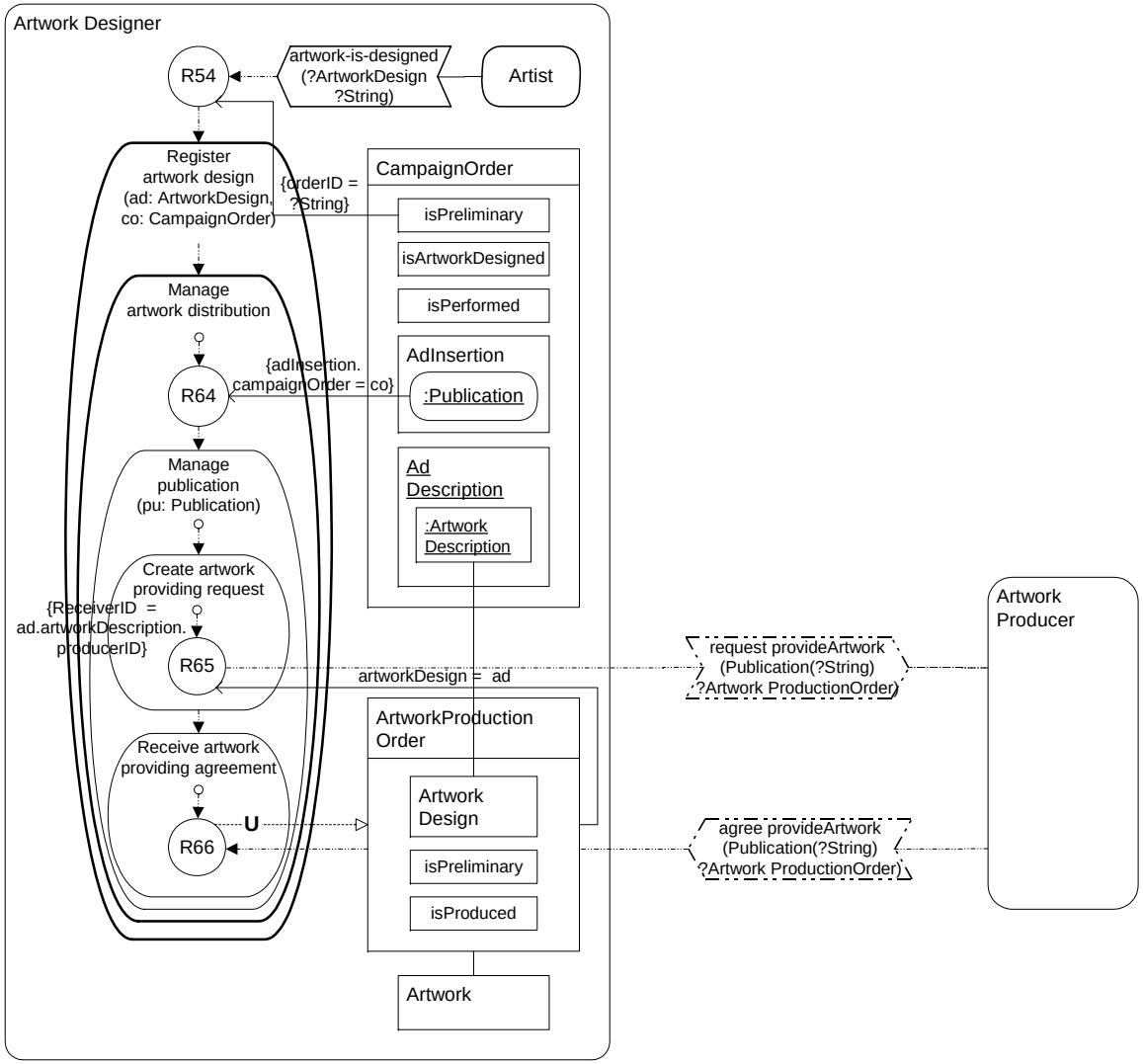
Appendix G (continued). AOR activity diagrams for the case study of advertising



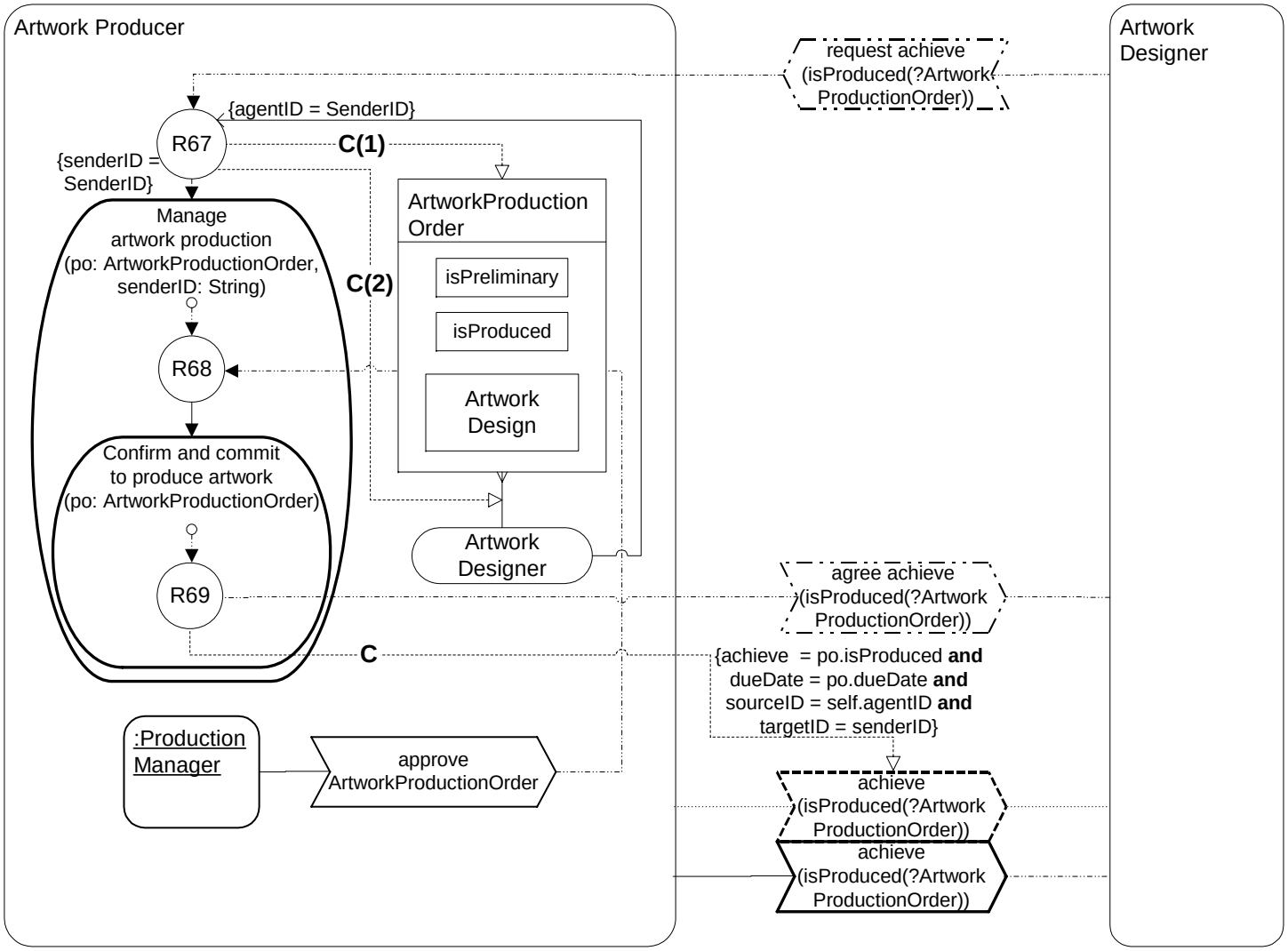
Appendix G (continued). AOR activity diagrams for the case study of advertising



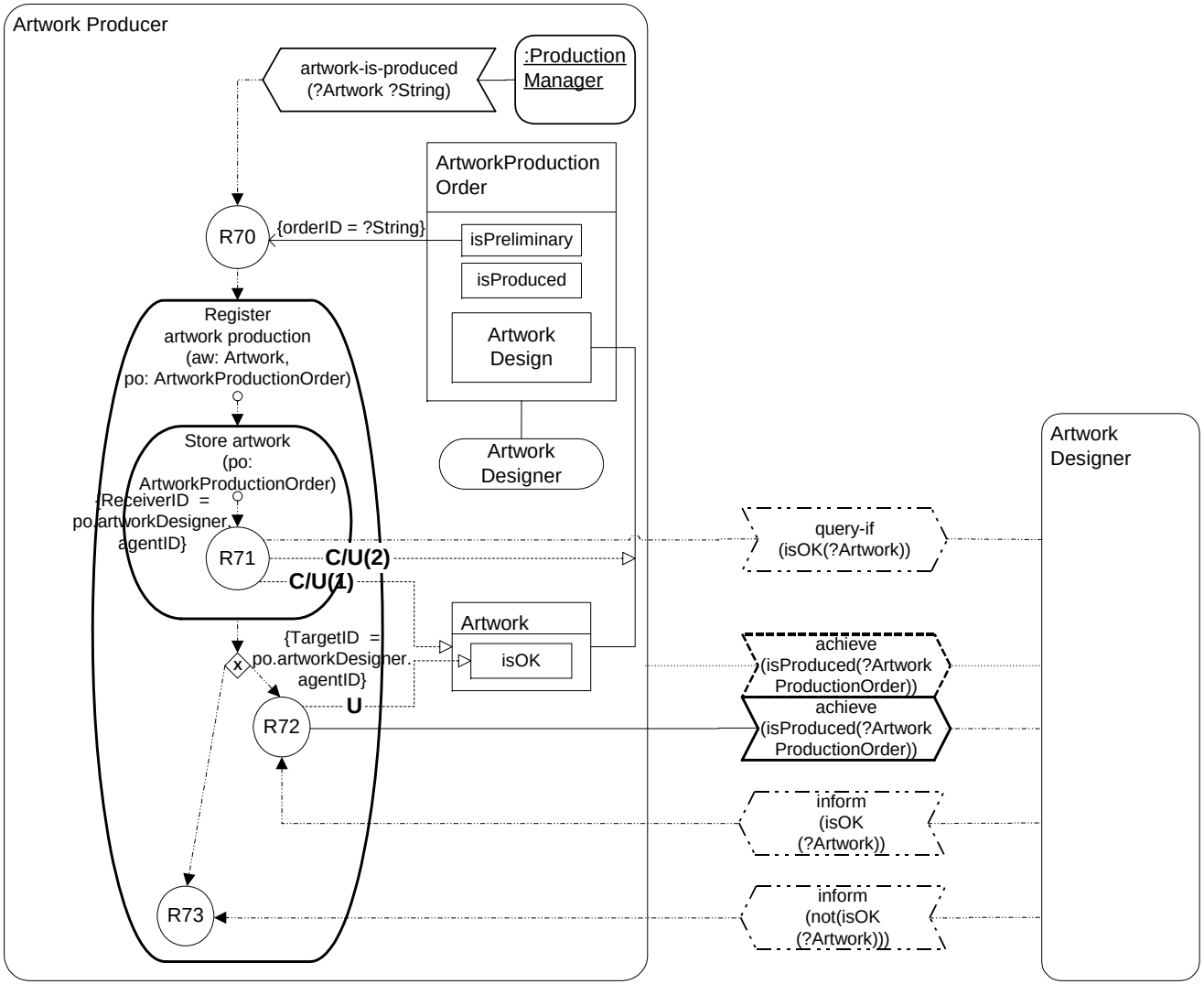
Appendix G (continued). AOR activity diagrams for the case study of advertising



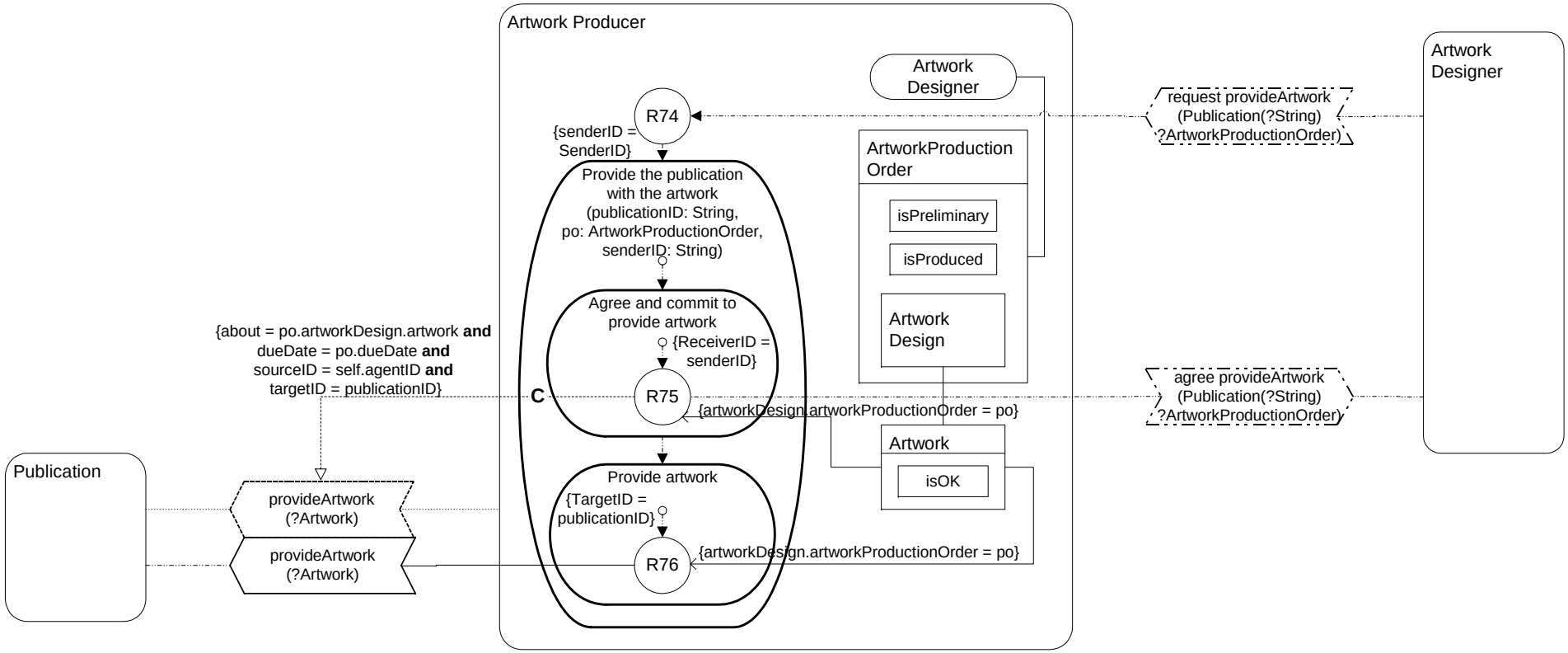
Appendix G (continued). AOR activity diagrams for the case study of advertising



Appendix G (continued). AOR activity diagrams for the case study of advertising



Appendix G (continued). AOR activity diagrams for the case study of advertising



Preliminary results of the research reported in this thesis have appeared in the following publications:

[Tamm96] Tamm, B., Taveter, K. A List-based Virtual Machine for COBOL. *Software – Practice and Experience*, Vol. 26 (12) (December 1996).

[Taveter97] Taveter, K. From Object-Oriented Programming Towards Agent-Oriented Programming. In: Grahne, G. (ed.), *Proceeedings of the Sixth Scandinavian Conference on Artificial Intelligence* (SCAI'97). Helsinki, Finland, August 18 – 20. IOS Press, 1997, p. 288.

[Taveter99a] Taveter, K. Java Agents that Realize Business Rules. *Proceedings of the 4th International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology* (PAAM'99). London, GB, 19 - 21 April 1999. Practical Application Company, London, 1999, pp. 471 – 472.

[Taveter99b] Taveter, K. Business Rules' Approach to the Modelling, Design and Implementation of Agent-Oriented Information Systems. *Proceedings of the 1st International Bi-Conference Workshop on Agent-Oriented Information Systems* (AOIS'99). Seattle, USA, 1 May 1999 and Heidelberg, Germany, 14 – 15 June 1999. Seattle, Heidelberg, 1999, pp. 59 – 75.

[Taveter00a] Taveter, K., Tamm, B. A New Approach to the Modelling, Design and Implementation of Business Information Systems. In: Barzdins, J., Caplinskas, A. (eds.), *Databases and Information Systems, 4th International Baltic Workshop, Baltic DB&IS, Selected Papers*. Vilnius, Lithuania, 1 – 5 May 2000. Kluwer Academic Publishers, Dordrecht, 2000, pp. 176 – 192.

[Taveter00b] Taveter, K., Wagner, G. Combining AOR Diagrams and Ross Business Rules' Diagrams for Enterprise Modelling. In: Lesperance, Y., Wagner, G., Yu. E. (eds.), *Agent-Oriented Information Systems 2000*. Proceedings of the 2nd International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2000). Stockholm, Sweden, 5 - 6 June and Austin, Texas, 30 July 2000. iCue Bublishing, Berlin, 2000, pp. 113 – 130.

[Taveter00c] Taveter, K. The use of deontic notions in agent-oriented business information systems. In: Hyötyniemi, H. (ed.), *Proceedings of the 9th Finnish Artificial Intelligence Conference* (STeP 2000). Espoo, Finland, 28 - 31 August 2000. Finnish Artificial Intelligence Society (FAIS), 2000, pp. 77 – 83.

[Taveter01a] Taveter, K. From Descriptive to Prescriptive Models of Agent-Oriented Information Systems. In: Wagner, G., Karlapalem, K., Lesperance, Y., Yu. E. (eds.), *Agent-Oriented Information Systems 2001*. Proceedings of the Third International Bi-Conference Workshop AOIS-2001. 28 May 2001, Montreal, Canada and 4 June 2001, Interlaken, Switzerland. iCue Publishing, Berlin, 2001, pp. 33 – 44.

[Taveter01b] Taveter, K., Wagner, G. Agent-Oriented Business Rules: Deontic Assignments. *Proceedings of the International Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations* (OES-SEO 2001). 14 - 15 September 2001, Rome, Italy.

[Taveter01c] Taveter, K., Wagner, G. Agent-Oriented Enterprise Modelling Based on Business Rules. In: Kunii, H. S., Jajodia, S., Sølvberg, A. (eds.), *Conceptual Modelling – ER 2001, 20th International Conference on Conceptual Modelling*. Yokohama, Japan, November 27 - 30, 2001. Proceedings. Lecture Notes in Computer Science 2224, Springer, 2001, pp. 527-540.

[Oja01] Oja, M., Tamm, B., Taveter, K. Agent-based software design. *Proc. Estonian Acad. Sci. Eng.*, 2001, 7, 1, pp. 5-21.

[Taveter02a] Taveter, K., Wagner, G. A Multi-perspective Methodology for Modelling Inter-enterprise Business Processes. In: Arisawa, H., Kambayashi, Y., Kumar, V., Mayr, H.C., Hunt, I. (eds), *Conceptual Modelling for New Information Systems Technologies, ER 2001 Workshops HUMACS, DASWIS, ECOMO, and DAMA*. Yokohama, Japan, November 27 - 30, 2001. Revised Papers. Lecture Notes in Computer Science 2465, Springer, 2002, pp. 403 – 416.

[Taveter02b] Taveter, K, Hääl, R. Agent-Oriented Modelling and Simulation of a Ceramic Factory. In: Juuso, E., Yliniemi, L. (eds.), *Proceedings of the 43rd Conference of Simulation and Modelling* (SIMS 2002). 26 – 27 September 2002, Oulu, Finland. Finnish Society of Automation and Scandinavian Simulation Society, 2002, pp. 102 - 110.

[Taveter04] Taveter, K. From Business Process Modelling to Business Process Automation. In: Cordeiro, J., Filipe, J. (eds.), *Computer Supported Activity Coordination – Proceedings of the 1st International Workshop on Computer Supported Activity Coordination* (CSAC 2004). In conjunction with ICEIS 2004, Porto, Portugal, April 2004. INSTICC PRESS, 2004, pp. 198 - 210.