Requirements Elicitation and Specification Using the Agent Paradigm: The Case Study of an Aircraft Turnaround Simulator

Tim Miller, Bin Lu, Leon Sterling, Ghassan Beydoun, and Kuldar Taveter

Abstract—In this paper, we describe research results arising from a technology transfer exercise on agent-oriented requirements engineering with an industry partner. We introduce two improvements to the state-of-the-art in agent-oriented requirements engineering, designed to mitigate two problems experienced by ourselves and our industry partner: (1) the lack of systematic methods for agent-oriented requirements elicitation and modelling; and (2) the lack of prescribed deliverables in agent-oriented requirements engineering. We discuss the application of our new approach to an aircraft turnaround simulator built in conjunction with our industry partner, and show how agent-oriented models can be derived and used to construct a complete requirements package. We evaluate this by having three independent people design and implement prototypes of the aircraft turnaround simulator, and comparing the three prototypes. Our evaluation indicates that our approach is effective at delivering correct, complete, and consistent requirements that satisfy the stakeholders, and can be used in a repeatable manner to produce designs and implementations. We discuss lessons learnt from applying this approach.

Index Terms—Agent-oriented software engineering, agent-oriented modelling, technology transfer

1 Introduction

Evidence suggests that incorrect and poorly-specified requirements are a major cause of software project failure, with two major contributing factors being requirements complexity and a lack of stakeholder input [9], [14]. Stakeholders are often not capable of articulating their requirements fully at the beginning of a project. Early-stage requirements tend to be imprecise, subjective, idealistic and context-specific [17]. In our earlier work [26], [34], we used an incremental approach to requirements modelling to engage stakeholders, acknowledging that requirements elicitation is better supported with an agile process that iterates the system design choices as it progresses. Instead of eliminating uncertainty early, we embrace it and withhold design commitment, at least until there is clarity and understanding between stakeholders of what it may mean to disambiguate [11]. Committing early to requirements can forgo an opportunity to properly disambiguate them [13]. The agent paradigm offers some benefits to requirements engineers, especially early in the requirements process. However, to continue to improve agent-oriented software engineering methods, experience building real systems with industry is needed.

Over the past several years, we have worked with several industry and academic partners, including Adacel Technologies,[1] Lockhard[2] [35], and Jeppesen[3] (this paper), to improve requirements engineering using agents as the central paradigm. Our industry partners face the problem of eliciting and recording requirements of complex systems that contain many interacting parts, and many interactions between different actors. One such product is a large-scale system for the air traffic domain that allows simulation of complex trade-offs between interacting actors in a socio-technical system. These partners have identified that the agent-oriented paradigm is a natural metaphor for modelling the social considerations in their systems, emphasising the "why" questions that can help in requirements elicitation, and producing models that are more accessible to their non-technical stakeholders [3], [26], [29], [41].

While existing agent-oriented requirements engineering approaches have matured over the past decade, our current industry partner identifies two major drawbacks with existing work, including our own:

1) Eliciting and recording relevant information in agent-oriented models is a non-trivial problem. Existing methodologies do not describe, in a systematic and prescriptive manner, how to elicit and record early-phase models. For example, Maiden et al. [21] state: "One problem that the published i^* framework does

- *T. Miller and B. Lu are with the Department of Computing and Information Systems, The University of Melbourne, Melbourne, Victoria, Autralia. E-mail: {tmiller, lbin}@unimelb.edu.au.*
- *L. Sterling is with the Faculty of ICT, Swinburne University of Technology, Melbourne, Victoria, Autralia. E-mail: lsterling@swin.edu.au.*
- *G. Beydoun is with the Faculty of Informatics, University of Wollongong, Wollongong, NSW 2522, Australia. E-mail: beydoun@uow.edu.au.*
- *K. Taveter is with the Institute of Informatics, Tallinn University of Technology, Tallinn, EU 19086, Estonia. E-mail: kuldar.taveter@ttu.ee.*

Manuscript received 4 Feb. 2013; revised 29 May 2014; accepted 11 June 2014. Date of publication 17 July 2014; date of current version 17 Oct. 2014. Recommended for acceptance by M. Jackson. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSE.2014.2339827

1. http://www.adacel.com/
2. http://www.lochard.com/
3. http://www.jeppesen.com/

not address directly is where to start i^* modelling". Thus, the modelling remains far more art than engineering. In our previous projects, we have observed experienced software engineers modelling the system in a way with which they are familiar, but using agent-related terms; for example, modelling the system using the object-oriented paradigm, but using "agent" in place of "object"; therefore negating any advantage of using the agent paradigm.

2) Existing methodologies do not prescribe how to define software requirements specifications (SRS) that incorporate agent-oriented models, while supporting traceability from requirements to design, implementation, verification, and back again. While traceability between models is a strength of existing agent-oriented methodologies, our industry partner could not see how to interpret agent-oriented models as requirements specifications that could be implemented and used for verification. Agent-oriented methodologies usually focus on the agent-oriented aspects, but overlook other aspects of software engineering, such as useful deliverables. Typically, a set of models is considered as a deliverable, providing little support for defining artifacts such as software requirements specifications, business vision documents, and system design descriptions, even though these are vital to industrial practice. This lack of support is unsurprising because publishing papers on deliverable formats may not be considered a valid scientific contribution.

As part of a larger grant funded by the Australian Research Council and Jeppesen, a subsidiary of Boeing that develops software for the aviation and aerospace industries, we are exploring how agent-oriented models can be used in conjunction with a mature piece of software that needs to be maintained and enhanced. In a previous article [26], we identified several techniques for engaging stakeholders in the requirements engineering process. We use lightweight, hierarchical agent-oriented models to represent the roles, goals, and motivations of the greater socio-technical system, and to develop a shared understanding of these goals between the project stakeholders. Further, we advocate several strategies for delaying design decisions with the aim of encouraging stakeholder involvement.

This paper offers the following two contributions that build on both our and other researchers' existing work in agent-oriented requirements engineering.

1) *Requirements elicitation, analysis, and modelling.* In Section 4, we present a systematic and repeatable approach for eliciting, analysing, and modelling the requirements of a system in an agent-oriented manner. This approach prescribes a list of questions that must be answered by the project stakeholders, and further prescribes how to link the answers directly to agent-oriented models. In particular, this approach aims to place requirements engineers into the "agent mindset", to gain the full benefit of agent-oriented modelling.

   Our approach improves on existing agent-oriented requirements elicitation approaches by providing a more prescriptive and systematic way for software engineers to elicit information and construct models from these.

2) *Requirements specification.* In Section 5, we prescribe how to create a requirements specification built on agent-oriented models, which emphasises those aspects of the agent paradigm that are important for understanding the system. The end result is not (necessarily) a specification of a multi-agent system, but a specification of a system that uses the agent paradigm to describe motivation, structure, behaviour, and interaction.

   Our requirements specification template improves on previous work in agent-oriented software engineering by advocating the inclusion of agent types in requirements specifications to describe behaviour.

To evaluate these contributions, we used our approach, embedded in the ROADMAP/AOR agent-oriented development methodology [34], to develop a complete requirements package for an aircraft turnaround simulator (ATS), in conjunction with our industry partner. We then tasked three people of varying experience and backgrounds, but with no previous training in agent-oriented software engineering, to each implement a prototype from the requirements package. An analysis and comparison of the three prototypes demonstrates that the our approach delivers requirements specifications that can be implemented in a repeatable manner, and does not require specialist knowledge. As our approach is centred around the concepts of roles, goals, and agents, we believe that agent-oriented methodologies built on similar modelling concepts could also be used.

The project and simulator are described in Section 2. An introduction to our previous work is discussed in Section 3. Applying our approach to the case study, we produced a requirements package that the client considered correct, complete, and consistent, and was developed into a prototype system (Sections 4 and 5). Section 6 presents the evaluation of our approach. Section 7 discusses the most important lessons that we learnt from the evaluation.

2 Case Study: Aircraft Turnaround Simulator System

In this section, we describe the running case study, built in collaboration with our industry partner, used in this paper.

2.1 The Research Project

This case study is part of a larger joint project between the University of Melbourne and Jeppesen, a company that specialises in aeronautical services. One of Jeppesen's flagship products is the *Total Airspace and Airport Modeller* (TAAM), which allows modelling and simulation of airports and the surrounding airspace to help with decision making regarding infrastructure and operations. This product is a large-scale complex system that contains many interacting parts, and many interactions between different actors. The current event-based model and implementation is proving difficult to understand, maintain and enhance due to its size and complexity.

Jeppesen identified that using agent-oriented methods may help to manage the complexity and scale of their

systems. The agent metaphor provides a natural and suitable way to represent a socio-technical system consisting of actors, their interactions, and their trade-offs. A major goal of the project is knowledge transfer between the University of Melbourne research team, and the software engineers at Jeppesen, as to how requirements should be elicited, modelled, and analysed.

As part of their own assessment of agent-oriented methods, Jeppesen identified that existing work does not provide a systematic and prescriptive method for eliciting requirements. Their engineers found it difficult to know where to start the process of developing agent-oriented models for requirements. In addition, during this project, it became clear that software engineers at Jeppesen could not see how agent-oriented models could be interpreted as requirements.

The core of the Jeppesen team on the project consists of three members with qualifications in physics, biophysics, and computer science respectively. All are familiar software engineering, but none have any significant prior experience in applying agent-oriented software engineering principles.

2.2 Aircraft Turnaround Simulator

As part of the knowledge transfer in the project, we have undertaken a smaller project, in which we aim to develop a simulator for aircraft turnaround using agent-oriented methods. This particular system was chosen because it is complex enough to demonstrate many aspects of the agent-oriented paradigm, but also manageable for our research group, who are not experts in aviation.

The system developed as part of the project is called the *Aircraft Turnaround Simulator* system. The ATS system simulates the process of multiple aircraft landing at a single airport, and how resources (including staff) could be allocated to efficiently turn around the aircraft, including re-stocking supplies, cleaning, repairing, and maintaining the aircraft. The intended usage is for Monte Carlo simulation of the aircraft turnaround process to evaluate different resource allocation mechanisms in airports.

The purpose of the system is to allow a user to evaluate different resource allocations mechanisms at airports. As such, the user should be able to set up parameters that specify the properties of the airport, the resources available, the schedules of staff, and the schedules of the arriving and departing aircraft. The system must produce reports describing the start and end points of all activities undertaken by staff, which can be used to assess the efficiency of allocation mechanisms.

3 Agent-Oriented Modelling

To model requirements, we use the notation of Sterling and Taveter [34]. Their work has focused on how to make high-level agent-oriented models palatable to non-technical stakeholders, and to carry these through to design and implementation. This is achieved using models with a straightforward and minimal syntax and semantics. In this section, we briefly describe the models that are relevant for this paper.

Like most related work on agent-oriented modelling, [3], [7], [40], [42], [44], the central concepts used in early requirements are goals and roles, rather than agents. These ideas have emerged because organisations and scenarios within them are better described as collections of roles and the goals they achieve, rather than specific individuals and tasks they undertake [40].

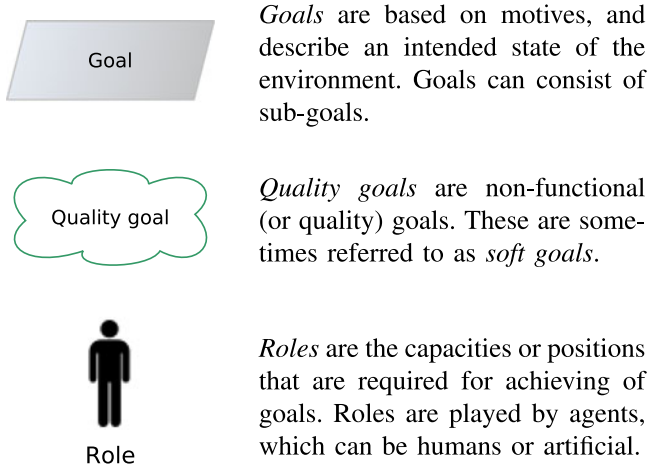

Fig. 1. Sterling and Taveter's notation for goal modelling.

Goal models are useful at early stages of requirements analysis to arrive at a shared understanding [15], [17]; and the agent metaphor is useful as it is able to represent human behaviour. Agents can take on roles associated with goals. These goals include quality attributes that are represented in a high-level pictorial view used to inform and gather input from stakeholders. For example, a role may contribute to achieving the goal "Release pressure", with the quality goal "Safely". We include such quality goals as part of the design discussion and maintain them as high-level concepts while eliciting the requirements for a system.

Role models describe the capacities or positions that facilitate the achievement of goals. Roles have *responsibilities*, which outline what an agent playing the role must do to achieve the related goals, and *constraints*, which determine the conditions that must be considered when trying to achieve goals. Fig. 1 defines the notation employed by Sterling and Taveter in their role and goal models. Goals are represented as parallelograms, quality goals are clouds, and roles are stick figures. These constructs can be connected using arcs, which indicate relationships between them.

Organisation models represent the relationships between roles in a system. Zambonelli et al. [43] define several relationships between pairs of roles, and these definitions are widely accepted in the literature. In our work, there are three relationships that we have found useful: *control*, in which one role delegates responsibilities to another; *peer*, in which either role can delegate responsibilities to another; and *benevolence*, in which a role offers to fulfil responsibilities for another if it is in the offering role's interests.

Domain models describe the relevant entities and relationships in the domain that the system operates. These can be represented in any suitable modelling language.

Agent and acquaintance models define the agents that will play the roles in the system, and the interaction pathways between the agents (similar to organisation models). Agents can be human or non-human, such as software or robotic.

Behavioural models and *knowledge models* specify the behaviours of the agents, and the knowledge that is required by the agents to perform their behaviours.

Interaction models represent communicative and physical interactions between the agents involved; that is, the activities in which two or more agents participate.

While the case study in this paper uses Sterling and Taveter's models, one can relate the approach to other agent-oriented notations and methodologies by identifying which models fit in the particular *viewpoints*. Sterling and Taveter [34] present the viewpoints of four other methodologies: Gaia [44], MaSE [7], Tropos [3], and Prometheus [29].

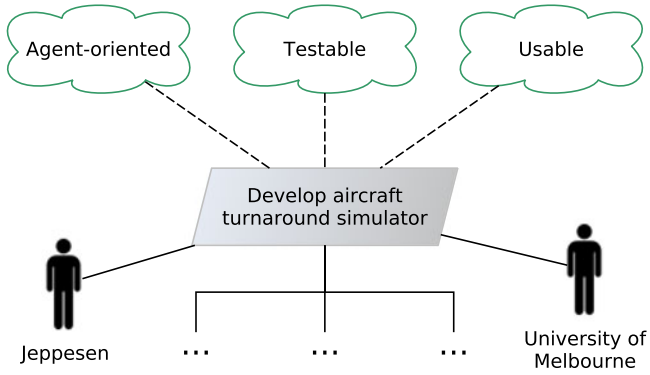

Fig. 2. An excerpt of the project motivation model for the ATS system.

4 Elicitation, Modelling, and Analysis

In our experience working with industry and academic partners, we have found that a major barrier to using the agent paradigm to engineer requirements is the mindset of the requirements engineer. For example, those people familiar with object-oriented modelling will naturally design an "agent" system in which agents are directly mapped to objects, and messages are directly mapped to method calls, thus eliminating any advantage of using the agent paradigm.

In this section, we propose an approach for agent-oriented requirements elicitation, analysis, and modelling as a series of questions aimed to identify what needs to be elicited, and to analyse the elicited information, producing agent-oriented requirements models of the system. The questions naturally lead the people answering them to think of the system in terms of roles, goals, and interactions—helping the requirements engineers to get into the "agent mindset".

It is important to note that these questions are not necessarily to be used as interview questions, although interviews can form part of the input. The questions form a checklist, but one in which items are posed as questions. These questions can be answered using techniques such as domain analysis, introspection, or group meetings. The questions and corresponding rules offer a prescriptive approach to producing models, and our experience is that most of the questions can be answered without having to ask them directly to a stakeholder.

The process followed is a straightforward elicitation process of identifying the problem and proposing a solution, involving the following steps:

Step 1. Identify the problem, root causes, and stakeholders.
Step 2. Develop a shared understanding of the *existing* system used to solve the problem, modelled using roles, goals, and interactions.
Step 3. Identify a solution that uses the metaphor of a new staff position solving the problem.
Step 4. Specify the agent types that will play the roles in the system, generally with the new staff position being partially filled by the new software.

4.1 Engaging Stakeholders in Elicitation and Modelling

In the ATS project, we elicited requirements using a combination of domain analysis, introspection, and round-table discussions with the stakeholders. These round-table discussions allow the models, and as a result, our understanding, to evolve over time. They are also one key to engaging the stakeholders, and to not committing to a design too early. We use the term "round-table" instead of "group meeting" to differentiate the standard process of requirements engineers asking questions and taking notes, to our process of the many different stakeholders deriving models during the discussions.

While some modelling was performed outside of these meetings, this was to produce models that could be used as a starting point for subsequent discussions, which were then modified in the round-table discussions.

4.2 Our Approach to Systematic Elicitation, Analysis, and Modelling

Our approach uses hierarchical abstraction to help deal with complexity in systems. We take a typical top-down approach of focusing on high-level details early in requirements engineering, and exploring the lower-level details once the high-level understanding is sufficient.

4.2.1 *Identifying the Problem, Root Causes, and Stakeholders: A Business Vision*

The first step is to identify the problem, the root causes of the problem, and the stakeholders. These properties of the project are recorded in what our industry partner terms a *business vision* document. The aim of this artifact is to reach a shared agreement of the problem, and also a high-level agreement of a solution space.

This step is standard in many projects, however, one difference to other approaches is that we use goal models to represent the motivations of the *project*, as well as the socio-technical system in which the software system will reside.

Fig. 2 presents the project motivation model for the ATS system. The goal of the two stakeholders is to develop an aircraft turnaround simulator. Three quality goals were noted. First, the product must be developed using the agent-oriented paradigm. While this may seem as a unnecessary constraint on the system design, it was an important *project* quality goal because the purpose of the project was knowledge transfer in the area of agent-oriented software engineering. Also, the product must be testable and usable. These are two important quality goals for all projects undertaken by our industry partner. In early conversations, measurable definitions of testable and usable were not

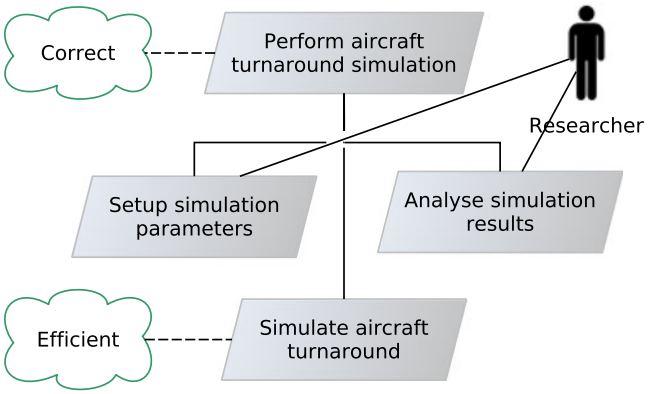

Fig. 3. Excerpt of the high-level motivation model for the ATS system.

important, but were refined and agreed upon in subsequent discussions.

In addition to the project motivation, we also derive a high-level model of the system motivation. This outlines the goals of the entire system, not just of the software to be built. An excerpt of this for the ATS system is shown in Fig. 3. This model identifies the key motivations of the researcher—the main user of the system, and how the system fits into their workflow. The quality goal *Efficient* refers to execution time, but as with other quality goals, the primary motivation at this stage is identifying the most important goals. Measurable definitions of quality goals are not always necessary at this stage, but should be defined before designing the corresponding part of the software.

These artifacts, including the motivation models, are signed off by the client (or major stakeholders). Our business vision documents have the structure outlined in Fig. 4.

Title information

Revision History

1 Introduction

2 Project Brief
- 2.1 *Problem*: description of the problem.
- 2.2 *Root causes*: root causes of the problem.
- 2.3 *Project stakeholders*: project stakeholders.
- 2.4 *Project motivation model*: project motivations.

3 Product Brief
- 3.1 *System overview*: overview of proposed solution.
- 3.2 *High-level product motivation model*: product motivations.
- 3.3 *High-level role models*: high-level system roles.
- 3.4 *Assumptions*: product brief assumptions.
- 3.5 *Constraints*: constraints on the solution.

4 High-level plan
- 4.1 *Project timeline*: high-level project timeline.
- 4.2 *Project deliverables*: list of artifacts to be delivered to the client.

5 Endorsement
- 5.1 *Sign-off*: between the client and the developers.

Fig. 4. A template for the business vision artifact.

4.2.2 Understanding the Current System

The second step is to understand the current system being employed to solve the problem; perhaps a manual system, or other software.

Zave and Jackson [45] argue the importance of understanding an entire system, including the environment in which a piece of software will operate. We agree that it is important to first understand the motivations of the existing socio-technical system, as any potential solution is likely to have the same motivations. This understanding includes all roles that are part of the system, and the goals achieved, whether these are achieved manually or otherwise.

Our approach uses high-level motivational scenarios of the current system to identify the roles and goals of this system by systematically stepping through the scenarios and answering a series of questions. Motivational scenarios are different to models such as use cases, in that they model interactions between the user and the software system *as well as* activities that do not cross the boundary of the software system. Scenarios can be derived by the stakeholders, or taken from existing artifacts. Unlike scenario-based requirements techniques [36], our scenarios can be highly unstructured. Our approach does not mandate any particular notation for scenarios, however, as a minimum, we require a set of high-level activities that occur in the system, and dependencies between these.

Fig. 5 shows a high-level motivational scenario for the ATS system. Motivational scenarios were provided by the client, and did not include or define any actors/agents.

The approach for eliciting information about the current system is to systematically step through every activity in the scenarios, one by one, and answer a series of questions about the activity, recording relevant information in models.

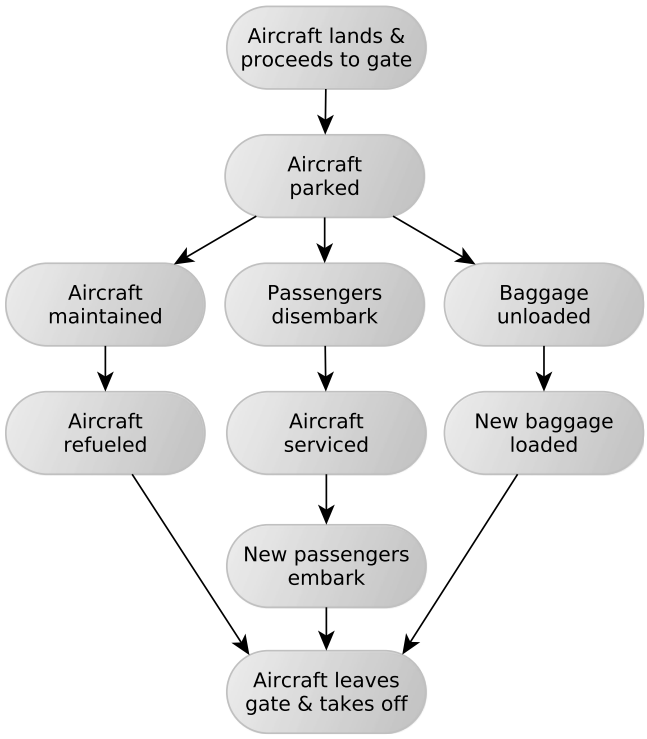

Fig. 5. High-level scenario used to elicit understanding of the aircraft turnaround domain.

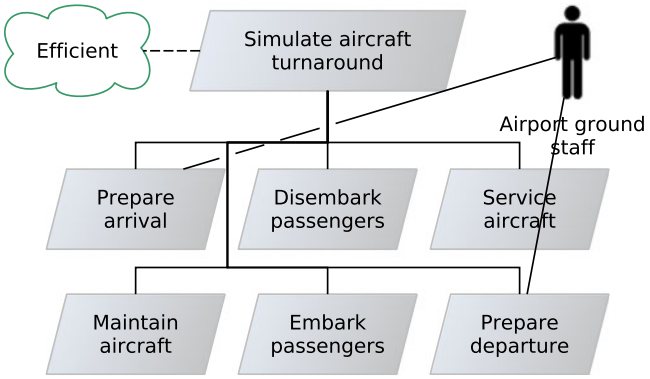

Fig. 6. The high-level motivation model of the aircraft turnaround process.

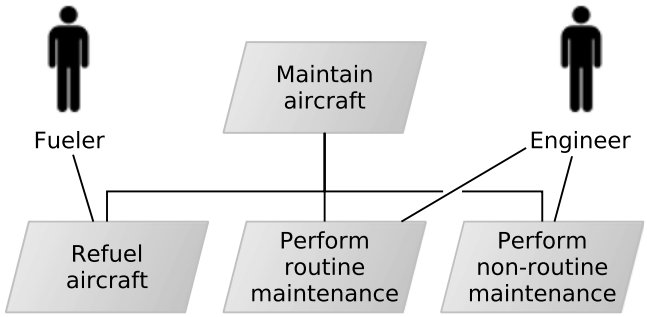

Fig. 7. The sub-goal of maintaining an aircraft during the turnaround process.

We identify the following questions for eliciting a complete set of models.

Q1 *What is the purpose of this activity?*
This question aims to elicit the goals and sub-goals of the scenario. Information elicited with this question is recorded in the goal model.

Q2 *Can this activity be broken into a set/series of smaller activity?*
This question aims to identify additional goals. If the answer is "yes", add this activity to the "stack" of activity to be analysed.

Q3 *Which roles take part in this activity?*
This question aims to elicit the roles in the system. These roles are recorded on the goal model.

Example. The aircraft turnaround process has a series of goals to achieve, which are achieved by roles. Fig. 6 shows the highest-level view of the aircraft turnaround process, including the major goals of the process. In our models, we typically annotate roles to either goals that are *leaf goals*—that is, are not broken down further—or to non-leaf goals in which the role is responsible for the non-leaf goal and its subgoals.

Once we have a sufficient understanding of the domain at a level, we investigate the lower levels in more detail, if required. Fig. 7 shows the excerpt from the motivation model for the *Maintain aircraft* goal from Fig. 6.

The maintenance of the aircraft ensures that the aircraft is safe to fly. There are two types of maintenance: routine and non-routine. Routine maintenance is performed by engineers after every flight. Non-routine maintenance is performed by engineers only if requested; for example, by the pilot, to investigate a potential problem. Only the engineers take part in the maintenance of the aircraft, so we add an *Engineer* role to the goal model. In a round-table discussion, we learnt that aircraft are not re-fueled by engineers, but by *Refuelers*.

The goal at top of the hierarchy in Fig. 7, *Maintain aircraft* is a leaf goal in the hierarchy of Fig. 6, and was expanded later in the requirements elicitation process. The hierarchical nature of the Sterling and Taveter's motivation models support this divide-and-conquer approach by allowing the inclusion of a goal in a high-level model, and then expanding that goal in new goal model later. Note that the roles responsible for the *Maintain aircraft* goal are annotated at the lower-level.

Q4 *For each role identified in Question 3*:

Q4.1 *If playing this role, which other roles would I rely on, and what are my relationships with these roles?*
This question aims to elicit the organisational and interaction relationships of the system. Information elicited with this question is recorded in the organisation model. This question also helps to identify other roles in the system. For additional roles, add them to the queue of roles to be analysed.

Q4.2 *What responsibilities would I have with respect to achieving the goal of this interaction?*
This question aims to elicit the responsibilities of the role. Information elicited with this question is recorded in the *responsibilities* attribute of the role model.

Q4.3 *What knowledge would I require to successfully complete this interaction?*
This question aims to elicit the knowledge required for an agent playing the role to successfully complete the interaction. Information elicited with this question is recorded in the environment model.

Q4.4 *What resources would I require to successfully complete this interaction?*
This question aims to elicit the relevant aspects of the environment. Information elicited with this question is recorded in the environment model.

Q4.5 *To which social policies (rules, regulations, or codes of behaviour) am I required to adhere to successfully complete this interaction?*
This question aims to elicit the constraints under which the role must operate. Information elicited with this question is recorded under the *constraints* attribute of the role model.

Q5 *Are there additional social policies to which participants in the scenario must adhere?*
This question further aims to elicit the constraints under which roles must operate. Information elicited with this question is recorded under the *constraints* attribute of the role model.

Example. The *Engineer* role relies on the *Pilot* role to inform it that non-routine maintenance should be

Role ID	R9
Role Name	Engineer
Description	The *Engineer* performs maintenance on the aircraft.
Responsibilities	1. Perform routine maintenance on a specified aircraft when informed of its arrival. 2. Perform non-routine maintenance on a specified aircraft when requested.
Constraints	1. Perform the routine and non-routine maintenance before the scheduled departure of the aircraft.

Fig. 8. The role model for the *Engineer* role.

performed, and on the *Manager* role to be instructed to allocate the staff schedule. The *Engineer* is a peer of the *Pilot* role, and is controlled by the *Manager* role.

The *Engineer* role is responsible for undertaking routine and non-routine aircraft maintenance, which must be completed before the aircraft can be re-fueled. Fig. 8 shows the role model for this role.

To fulfil their responsibilities, an *Engineer* is required to know the aircraft ID, the gate number at which the aircraft is parked, and that the air-bridge has been positioned. The resources required are the flight plan, staff schedule, and aircraft information. The physical resources are the aircraft itself and the maintenance equipment. This information is recorded in the environment model.

4.2.3 Eliciting a Solution: Hiring New Staff

To elicit a solution for the problem, we build on the HOMER elicitation technique proposed by Wilmann and Sterling [39], which uses the metaphor of hiring staff in an organisation. The stakeholders are prompted to consider how their problem could be solved by hiring new staff members, perhaps by temporarily ignoring some quality goals; e.g., a quality goal of *Efficient*, referring to the time taken to complete a task, may not be achievable using a human. For non-technical stakeholders, this metaphor is an intuitive way to conceptualise a solution, and for technically-minded stakeholders, this metaphor forces them to think more about the *"how?"* and *"who?"* aspects of the system, rather than just the *"what"* aspect.

The questions used to elicit the motivations of the new system are:

Q1 *If one was to hire more staff to handle the problem, what positions would you need to fill?*
This question aims to elicit the new roles that will be added to the system. Information elicited with the question is recorded as roles on the goal model.

Q2 *For each new role identified in Question 1*:

- Q2.1 *If playing this role, what is the purpose of my position, and what aspects of the problem would I solve?*
This question aims to define any new goals or sub-goals in the system. Information elicited with this question is recorded in the goal model.
- Q2.2 Ask Questions 4.1-4.5 from Section 4.2.2.

Q3 *Are there any new social policies to which I must adhere?*
This question aims to elicit any new constraints under which roles must operate. Information elicited with this question is recorded under the *constraints* attribute.

Example. In the turnaround project, new staff could be hired to perform human-based simulations of the turnaround process, allowing evaluation of different resource allocations. Clearly, we were aware that the simulation would be software-based, so we elicited the roles of the system with the knowledge they would be implemented as software agents.

For the purpose of illustration of how this step would work in a non-simulation, we consider an alternate—but closely-related—system, in which we are modelling the ATS in order to add a new re-fueling system to an airport. In this alternative system, an agent fulfilling the *Engineer* role is responsible for re-fueling the aircraft, and there is no *Fueler* role.

The problem that the organisation encounters is that the turnaround is being delayed by the *Engineer* being unable to perform the routine maintenance and refuel the aircraft quickly enough, delaying take off. To solve this problem, the stakeholders identify that they could hire a person to specifically refuel the aircraft in parallel with the engineer performing routine maintenance.

One potential problem is that the staff-hiring metaphor could shift the mindset of the stakeholders outside of the software mindset all together, resulting in a sub-optimal solution. However, in practice, stakeholders in a project have preconceived ideas about technical solutions, so would likely be aware that the solution can be software based, hardware based, human based, or a combination. In our experience, we have not encountered this problem.

4.2.4 Defining the Solution: Deciding the Agent Types

To define the solution, we must define two things: 1) the software system boundary, which is the boundary between the software and its users and environment; and 2) the behaviour of the software that will solve the problem. We define both of these by specifying the agent types that will play the roles in the system.

Cheng and Atlee [4] advocate defining a system boundary by assigning responsibilities to different parts of the system, such as the software system being constructed, human operators/users, and external systems. We do the same by specifying which roles will be played by human agents, by external systems (either hardware or software), and by software agents.

For example, one possible boundary discussed with the stakeholders in the ATS project was to have the *Manager* role played by a human, instead of a software agent. By changing this assignment of roles, the system changes from a constructive simulator into a human-in-the-loop simulator.

To elicit the behaviour of the system, ask the following questions for each responsibility identified in each role model:

Q1 *If playing this role, what activities would be required for me to fulfil my responsibility?*

This question aims to elicit the behaviour of the system that will fulfil the given responsibility, and contribute to achieving to the system goals.

Q2 *For each activity identified in Question 1*:

- Q2.1 *Is this activity performed by a human agent, an external system, or a software agent?*
  This question aims to assign responsibility to the parts of the system.
- Q2.2 *If performing this activity, what would I have to do?*
  This question aims to break an activity down into sub-activities and atomic actions. Actions are activities that are not considered in further detail. Information elicited with this question is recorded as information about the structure of the corresponding activity in the activity register.
- Q2.3 *What help would I require from other agents to successfully complete this activity?*
  This question aims to elicit possible messages that may be sent and received to complete this activity. Information elicited with this question is recorded in the interaction model.
- Q2.4 *What prompts me to undertake this activity?*
  This question aims to elicit the trigger for the activity; that is, the event to which the agent reacts. Information elicited in this section is recorded as the trigger of the corresponding activity in the activity register. If this trigger is a message received from another agent playing some role, it is also recorded in the interaction model.
- Q2.5 *Under what conditions can I undertake this activity?*
  This question aims to elicit the precondition for the activity; that is, the states of the environment that enable this activity. Information elicited with this question is recorded as the precondition of the corresponding activity in the activity register.
- Q2.6 *What happens after I complete this activity?*
  This question aims to elicit how the activity changes the environment. Information elicited in this section is recorded as the postcondition of the activity.
- Q2.7 *What other agents do I need to inform that this activity has been completed?*
  This question aims to elicit information regarding interactions, and the actions for this activity. This information is recorded in two places: 1) as a part of high-level description of the actions for this activity (as in Question 2.2); and 2) in the interaction models.

Example. In the ATS system, the routine maintenance will be performed by a software agent, and is modelled as an atomic action for simulation purposes. To perform routine maintenance, the aircraft must have its wheel chocks positioned, and the activity is triggered when the *Engineer* is informed by *Airport Ground Staff* that the aircraft is ready for maintenance. After the routine maintenance activity is complete, the aircraft is in a state in which the *Engineer* deems that it is safe to fly, and it informs the *Pilot* of this.

Activity name:	Routine maintenance.
Trigger:	Informed by *Airport Ground Staff* of the aircraft ID of the aircraft that is ready for maintenance.
Precondition:	Wheel chocks of the aircraft ID are in position.
Sub-activities:	1. Perform the routine maintenance on the specified aircraft. 2. Inform *Pilot* that the routine maintenance is complete on the aircraft.
Postcondition:	Aircraft with the specified ID is safe to fly.

Fig. 9. Activity description for *Routine maintenance* activity.

The final activity description for the routine maintenance is shown in Fig. 9.

The final step is to decide how the agent types will be defined to play the roles, and to perform the activities. We do not define a solution for this in this paper, as there are a number of useful methodologies that define this [29], [34], [44]. In our experience, agent types can be defined as a one-to-one mapping for simulation systems; that is, each role is mapped to one agent type, with the activities relating to a responsibility also mapped to the agent. In the ATS system, we used a one-to-one mapping, which was intuitive and clear from the role definitions, and is likely to be similar for any simulation system. However, there will be cases in which a one-to-one mapping is not possible. One clear case is when the responsibilities defined by a single role are mapped to multiple activities, but the activities are split over different categories; e.g., some are identified as being played by a human agent, and some by a software agent.

An an example of an agent type, consider the *Engineer* agent, which fulfils the role of *Engineer*. The analysis of the activities results in the definition found in Fig. 10.

The end result of the process is a set of agent types, which fulfil the system roles, and a description of the behaviour of these agents, specifying what the agent does, and how this affects the environment.

5 Specification and Packaging

The authors are unaware of any agent-oriented methodologies that provide support for creating deliverables such as software requirements specifications, even though these deliverables are a vital part of many software engineering projects. In this section, we present the definition of a *software requirements specification*—a deliverable describing the software system to be built. An SRS defines a shared understanding between the project stakeholders as to what is to be built; thus, it is an agreement between these stakeholders.

5.1 Describing Socio-Technical Systems Using Agent-Oriented Models

Using the organisational metaphor to elicit the requirements of a socio-technical system is beneficial, and we extend this to the specification of the system as well.

Name:	Engineer
Description:	Play the role of *Engineer* by performing routine and non-routine aircraft maintenance.
Activities:	**Activity name:** Routine maintenance
	...
	Activity name: Non-routine maintenance
	...
Environment considerations:	1. Aircraft 2. Aircraft information 3. Flight schedule 4. Aircraft gate number 5. Staff schedule

Fig. 10. Agent type specification for the *Engineer* agent.

To specify the system, we adopt the systems theory view, acknowledging that organisations and socio-technical systems are systems in their own right. The view of a system is broken in structure, behaviour, interaction, and purpose. The purpose of a system influences its structure and behaviour, which also influence each other, and the interactions. To understand a system, one cannot view any of these aspects in isolation; they must be considered as a whole. Fig. 11 shows where Sterling and Taveter's models reside with regarding to the systems view.

Many agent methodologies consider the definition of agents as a design step (e.g., see [44, Fig. 6]), and hold the view that including this in the specification unnecessarily constrains design (see [34, Ch. 6]). Using the models described in Section 3, this would imply that a specification consists of role models, organisational models, environment models, goal models, and motivational scenarios. It is our view that these models do not provide the "what" required to produce an unambiguous software requirements specification.

We advocate the inclusion of (at least some of) the models in the platform-independent computational design [34]. Those models that we deem necessary to include are the *agent type* specifications, which determine the behaviours of the agents in terms of the activities that they perform (presented in Section 4.2.4), and *agent models*, which determine the instances of each agent type, and which roles each instance plays. Additionally, we believe that it would prove valuable to include the interaction models, which determine the protocols between agents, and the knowledge models that are used for representing the agents' knowledge. Other agent-oriented methodologies could be interpreted in a similar manner.

5.2 Packaging the SRS

Producing a complete SRS requires us to package these models together in a meaningful way. Fig. 12 presents a possible template for an agent-oriented SRS, based on existing templates such as Wiegers' SRS template [38] and the IEEE Standard for requirements specifications [16]. Using a template leads to requirements being presented in a consistent manner across different projects, however, we acknowledge the need to be flexible with specifications depending on the system.

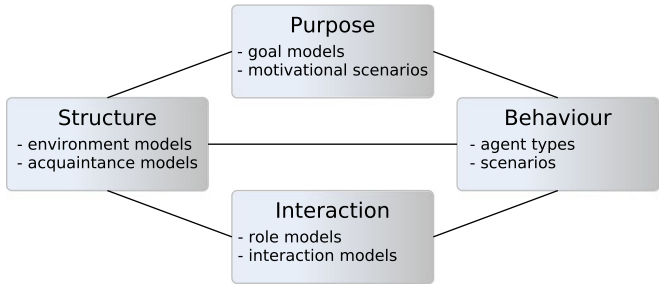

Fig. 11. The structure of socio-technical systems, as viewed from the agent paradigm.

One can see from Fig. 12 that the SRS is not structured the same as Fig. 11. Instead, the specification is presented based on abstraction, with higher abstraction being presented earlier. The scenarios in Section 7 of the template refer to those scenarios at the platform-independent design layer, so are presented later rather than earlier.

Sections 1 and 2 of the template are similar to that of Wiegers' [38], except that Section 2 includes the high-level motivational model (e.g., Fig. 3), and Section 2.4 of Wiegers' template (Operating Environment) has been considered as its own section (Section 5), emphasising the importance of the environment in socio-technical systems. Wiegers' *System Features* section has been replaced by the agent types and interaction models (Section 6), as this defines the behaviour of the system. Sections 3 and 4 in our template have no equivalent in Wiegers' template, as Wieger does not consider the purpose of the system and its roles. Wiegers' *Other Nonfunctional Requirements* section is not included, as nonfunctional requirements are considered as quality goals in the goal models.

5.2.1 A Note on External Interfaces

Section 8 of the template in Fig. 12 advocates specifications of the external interfaces of the system. To identify the interactions that are involved in each of these interfaces, we consider the organisational model and the agent types.

Recall that in our approach, agents are classified as human agents, software agents, or external hardware/software systems. We identified that any *new* software agents would be part of the software system that is to be implemented. Therefore, by analysing the interactions between these new agents and the other agents in the system, we can determine where the external interfaces will be, and use this to feed into the interface specification.

The categorisation of the agent types provides a clear divide for each interface type:

- interactions between a software agent and a human agent will take place at the user interface;
- interactions between a software agent and an external hardware or software system will take place at the hardware or software interface respectively; and

Title information

Revision history

Table of contents

1 Introduction
- 1.1 Purpose
- 1.2 Intended audience
- 1.3 Project scope
- 1.4 Definitions, acronyms, and abbreviations
- 1.5 References

2 Product Description
- 2.1 High-level level motivation model
- 2.2 User classes
- 2.3 Product features
- 2.4 Design constraints
- 2.5 Assumptions

3 Goal models and motivational scenarios
- 3.1 Motivational scenarios
- 3.2 Goal models

4 Role and organisational models
- 4.1 Organisational model(s)
- 4.2 Role 1
- 4.3 Role 2

etc ...

5 Environment model
- 5.1 Physical environment
- 5.2 Virtual Environment
- 5.3 Environmental perspective
- 5.4 Overall interaction

6 Agents types and interaction models
- 6.1 Interaction models
- 6.2 Agent type 1
- 6.3 Agent type 2

etc ...
- 6.4 Agent models

7 Scenarios
- 7.1 Scenario 1
- 7.2 Scenario 2

etc ...

8 External interfaces
- 8.1 User interfaces
- 8.2 Hardware interfaces
- 8.3 Software interfaces

9 Endorsement
- 9.1 Sign-off

Fig. 12. A software requirements specification template based on the elicited models.

- interactions between a human agent and an external hardware/software agents fall outside of the software boundary, so are not considered.

6 Evaluation

In this section, we discuss the activities undertaken to validate our requirements specification, therefore evaluating its strengths and limitations. To evaluate our approach, we undertook three distinct activities:

1) Technical reviews were performed by an individual who was not part of the requirements team.
2) Cross validations of the the agent-oriented models against an ontology derived were undertaken by an individual who was not part of the requirements team.
3) Three prototypes of the system were implemented by three different developers, each of differing experience, who were not part of the requirements team.

These three activities were used to identify problems in our approach, highlighting some problems and allowing us to refine several areas.

6.1 Reviews

Informal technical reviews were undertaken during roundtable meetings with our industry partners as part of the elicitation process. Once we had a set of stable models, these were reviewed by a person external to the requirements team, who had knowledge of the turnaround process from the documents provided to us by our industry partner, but had no access to the data derived using our approach.

The goal of the reviews was to locate defects in the requirements specification. First, the external reviewer performed a technical review based on their understanding of the aircraft turnaround domain, identifying any potential defects in the requirements specification. Second, the requirements team analysed the list of potential defects, identifying which were considered true and false positives respectively. Finally, the requirements team categorised each defect as either *minor* and *major*. A defect is considered major if it may lead to a defect in the resulting software; e.g., inconsistencies between a trigger for an agent's activity and corresponding scenarios, or responsibilities in a role not fulfilled by an agent playing that role. All other defects are considered minor; e.g., typos, small inconsistencies between models.

The reviews were performed iteratively in three stages. The first review was performed on an early version of the requirements specification that consisted of only roles, goals, environment, and some scenarios. The second and third review were performed on more complete versions that also included agent types and all scenarios. All reviews were undertaken by the same person. The third review uncovered no new defects, so data for this is omitted.

Table 1 shows the results from the first two reviews. No major defects were located in the first review, however, four were located in the second. There was a large increase in the number of minor defects located in the second review; from two to 21. While there were additional models reviewed in the second review, analysis of the data showed that the reason the number is so high is that the reviewer was able to cross-validate models against each other. That is, many of the defects found in the second review were inconsistencies between the agent types and the role models, and between the agent types and the scenarios. For example, two of the major problems in the

TABLE 1
Defects Located by Technical Review

	Iteration 1		Iteration 2	
Model	Major	Minor	Major	Minor
Goal	0	0	0	0
Role	0	1	2	6
Env	0	1	1	4
Agent	-	-	1	7
Scenario	-	-	0	4
All	0	2	4	21

agent types were incorrect triggers on action, which were found by noting that the triggering event did not occur before the action in the corresponding scenario.

6.2 Cross-Validation of Models

In previous work [18], three of the authors proposed a cross-validation method for agent-oriented models using ontologies. The basic premise is that an ontology is derived independently of the requirements specification, and then compared against the agent-oriented models for consistency. The ontology models the important concepts and relationships between the concepts in the system. Any inconsistency between the two indicates a problem with either the ontology or the models. In this section, we overview this and discuss some of the results.

The ontology was developed based on documentation from the client and some information from additional sources. The team deriving the ontology worked independently from the other members in the requirements engineering phase to introduce diversity between the models and the ontology. Throughout the development, the ontology was revisited to ensure that it is updated with any additional insights the client develops through interacting with the development team. The ontology consists of 350 concepts and relations. In addition, the ontology is augmented with annotations describing concepts and relations specifically relevant to agent-oriented models. Table 2 illustrates these properties for the models at the conceptual domain modelling level. These properties provide software engineers with a straightforward way to evaluate the consistency between the ontology and the other models.

The validation process consists of applying, for every model developed, a list of rules. Applying an rule can trigger one or two proposals to amend the model, depending on the outcome. For example, for validating our model, ten operators can trigger amendment proposals, exemplified by the following: "*add agent type X to the model set*", where *X* is defined in the ontology but it does not have a corresponding model. To ensure effectiveness of the cross-validation activity, the creators of the models were not directly involved in the validation. Instead, as the requirements engineering was undertaken, the modellers received recommendations from the team members undertaking the cross validation. Iterations were undertaken until models converged and no further amendments were proposed by the validation activity. Additional details can be found in earlier work [18].

TABLE 2
Ontology Properties

Domain	Property	Range
Goal	*is a*	Goal
Goal	*part of*	Goal
Role	*responsible for*	Goal
Role	*participates in*	Activity
Role	*is peer*	Role
Role	*controls*	Role
Role	*is benevolent*	Role
Role	*uses*	Environment
Agent	*plays*	Role
Agent	*performs*	Activity
Activity	*fulfils*	Goal
Activity	*requires*	Environment
Activity	*precedes*	Activity
Activity	*follows*	Activity

The cross-validation activity followed the same process as reviewing: identify potential defects; eliminate false positives; and categorise the remaining defects as minor or major. As with the reviewing, cross-validation was undertaken on three iterations by the same person, and no defects were located in the third iteration. It is important to note that the cross-validation was performed after the technical review in each iteration, and only previously undetected defects were recorded.

Table 3 shows the results from the two cross-validation activities. As with the results from the reviews (Table 1), more defects were located in the agent and scenario models than other models. However, in the cross-validation process, new defects were only found in new models. As with the reviews, the three major defects found in the agent types were all inconsistencies with the scenario model, meaning that a developer could implement the incorrect behaviour from one of these models.

6.3 Prototyping

As part of our evaluation, three different developers of varying experience were asked to design, implement, and test prototypes of the aircraft turnaround system. We used their resulting prototypes and their experience to evaluate whether a range of people can use, understand, and implement systems in a repeatable manner from our requirements specifications—something that is a fundamental concern for our industry partner. One prototype (developed

TABLE 3
Defects Located by Cross-Validation

	Iteration 1		Iteration 2	
Model	Major	Minor	Major	Minor
Goal	0	0	0	0
Role	0	0	0	0
Env	1	1	0	1
Agent	-	-	3	3
Scenario	-	-	0	4
All	1	1	3	8

during the requirements elicitation) was additionally used as a requirements elicitation and validation tool.

As part of the project, we asked three software engineers of varying backgrounds to design, implement, and test prototypes. Their respective details are:

1) a third-year undergraduate software engineering student at the University of Melbourne, Australia, with no previous experience in agent-oriented software engineering;
2) a coursework Master's student in software engineering from the Tallinn University of Technology, Estonia, who had worked as an air-traffic controller and had no previous experience in agent-oriented software engineering; and
3) a visiting scholar to the University of Melbourne, Australia, who is a software engineer with a master's degree and over ten years experience specialising in software for air-traffic control, but with no previous experience with agent-oriented software engineering.

6.3.1 Elicitation and Validation

The first prototype was produced during the requirements engineering process, and thus iterated along in tandem with the requirements specification. This prototype was used to validate the requirements against our industry partner's needs, as well as to elicit new requirements.

The feedback on the model from the student writing the first prototype was important for us as he had not been a part of the project beforehand, so therefore he gave an external viewpoint. It is interesting to note that the student commented on the ease of using the SRS. Although he was provided with several references on understanding agent systems and a textbook on Sterling and Taveter's models [34], the student commented that these were largely unnecessary because the agent-oriented models were straightforward to interpret.

Over two meetings, the scenarios from SRS were walked through with our industry partner, who highlighted a handful of incorrect assumptions that we had made, and noted some pieces of information that were missing from the SRS.

Due to the round-table nature of the discussions, a complete list of problems found was not recorded—instead, models were updated in the meeting based on the prototype feedback. However, the two major problems discovered included that the simulation should be able to be sped up and slowed down (it had been set to run at only one speed), and that manager role should be more tightly integrated with some events notifying agents of activities being completed, so that it could be kept informed of which agents and resources were free.

At the end of the second meeting, our industrial partner was confident that the final SRS was correct and complete with respect to their needs.

6.3.2 Comparison of Prototypes

Goals. The three major evaluation goals are to determine whether:

1) our elicitation method results in models that capture all requirements correctly and completely;
2) our SRS template is comprehensive enough to represent requirements; and
3) our approach results in *usable* requirements specifications, which developers can be used by developers without any specialised knowledge, to design and implement software in a repeatable and traceable manner.

While such properties are somewhat subjective, our hope was that a controlled comparison between three prototypes would reveal any shortcomings in our approach that could be improved upon.

Method. To achieve the three goals, the comparison consisted of the following steps:

1) analyse the associated artifacts (design documents, etc.) that were produced by the developers to assess consistency and traceability;
2) analyse the source code, and input and output spaces of the three prototypes to assess consistency and traceability; and
3) run several basic tests on the three prototypes to assess correctness, completeness, and consistency.

Finding that the designs and implementations are correct, consistent, complete, and traceable provides evidence that the SRS is a useful deliverable for describing the aircraft turnaround simulator.

Objects. Table 4 contains information characterising the three different prototypes, in which #Agents, #Modules, and LOC refer to the number of agents, number of modules/classes, and lines of code respectively.

TABLE 4
Metrics for the Three Prototypes

	Language	#Agents	#Modules	LOC
1	AgentSpeak	11	32	2872
2	JADE/Java	6	25	5051
3	JADE/Java	11	31	6677

Results. In this section, we highlight some of the important factors that we learnt from the comparison.

Designs. The designs produced by the three developers were closely consistent with each other. Developers 1 and 3 produced similar designs, which both opting for a straightforward one-to-one mapping between agents and roles, and similarly for mapping to agent behaviours and environmental entities. The design produced by developer 2 consisted of only six agents, with five of the roles from the SRS (*Fueler*, *Engineer*, *Airport Ground Staff*, *Airline Catering Staff*, *Airline Cleaning Staff*) being combined into a single agent type called *Airport staff*, and *Passenger* being removed all together, as the only activities, embarking and disembarking, were simulated as simply a time period between the bordering events.

This provides evidence that the designers considered the ATS SRS as a usable artifact, even though developer 2's design indicates that they may have combined roles in the SRS as well as in their design.

It is important to note that two different languages were used. Prototype 1 was implemented using AgentSpeak [32] (we used the Jason tool to interpret the source[4]), while prototypes 2 and 3 are written using Jade/JAVA.[5] This demonstrates that the SRS is independent of any particular platform.

Traceability. Due to the straightforward mappings used in prototypes 1 and 3, tracing both forwards and backwards between SRS and source code is achieved in a trivial manner. The slightly different mapping used in prototype 2 is still straightforward to trace, as the mapping between the roles and agents is clear, and the corresponding agent activities in the SRS are grouped sensibly in the prototype design.

Tests. Due to the ATS being used for Monte Carlo simulation, the times taken to complete activities are randomised to specified probability distributions. This makes it difficult to accurately compare the behaviour of the three prototypes using a test suite. Similarly, the emergent nature of the system means that complicated scenarios are difficult to verify. Instead, we ran six baseline tests that were straightforward to compare, such as a single aircraft turnaround with the optimal number of resources, which should therefore be simpler to validate.

The test results indicate consistency across the three prototypes when running these tests, and importantly, the test outputs were consistent with our expectations of the behaviour.

However, one problem that we noted during testing was the inconsistency of input setup for the three different prototypes. While the input domains were consistent, the way in which input was constructed varied significantly, to the point where learning how to run scenarios required a non-trivial amount of effort. Our analysis led us to the conclusion that this was the result of a weakness in our approach.

In Section 5.2.1, we describe how the organisational model can be used to specify external interfaces to the system, including the user interface. However, what was lacking in our approach was a mapping from these "interface" agents, to the input space of the simulator. As such, the developers made their own decisions as to how these should be done, which resulted in sensible but inconsistent prototypes.

Overall. Addressing our evaluation goals from earlier in this section, the evaluation noted the following results:

1) our elicitation method, in conjunction with prototyping, was successful in producing a correct and complete set of requirements;
2) our SRS template was incomplete, missing out the important relationship between the input domain and the behaviour specified by the agent types, but otherwise represented the requirements comprehensively; and
3) the prototypes provide evidence that our approach produces usable requirements specifications that can be implemented in a repeatable manner, and the ability to trace the all three designs and implementations back to the SRS demonstrates that the SRS supports tracing.

6.4 Summary

From our three methods to evaluate the ATS specification, the following key results are noted:

1) the reviews and cross-validation revealed eight major defects in the specification, mostly related to inconsistencies between agent activities and scenarios;
2) comparison of models showing different views found a majority of the defects in the SRS;
3) the link between the system input domain and the agent activities, which define the behaviour, was not clear enough.

7 Lessons Learnt

In this section, we discuss the most important lessons that we learnt as part of the project, about our method and about agent-oriented requirements engineering in general.

7.1 Requirements Elicitation

Business vision documents. Business vision documents are typically not required for academic projects, and therefore have not been addressed in the agent literature as far as the authors are aware. The use of high-level models in such documents is a novel application of agent-oriented models that was significant for our industry partner. The business vision models were helpful in identifying the high-level motivations and the stakeholders. For example, within seconds of being presented the first draft of the model in Fig. 3, one of the industry partners noted that the air traffic controllers were stakeholders in the turnaround process, and this induced discussion about how new traffic enters the airport. In subsequent iterations, the air traffic controller role was deemed unnecessary for the simulation and was dropped, but changes related to this remained. It is not unreasonable to claim that if the scope of the system was larger, we would have been required to engage with air traffic controllers as part of the elicitation process, and we believe our model would have identified this earlier than otherwise. Events such as this further strengthen our view that using stakeholders as modellers is highly valuable.

Scenarios. The key highlight of the project for our industry partner was the use of scenarios. High-level scenarios are an important starting point in our approach, and are useful in early phase elicitation and modelling. While scenarios have been applied in software engineering for many years [36], their use as a tool for both elicitation *and understanding* was a major finding for our industry partner. In a meeting after the ATS project had concluded, a team member from our industry partner described a more recent project in which they were contracted to build a simulation of small aircraft flights around an airfield. In the past, the company would have sent a business analyst to elicit requirements that were then passed on to software engineers. However, as a result of their insight from the ATS project, they additionally sent the team of software engineers

4. See http://jason.sourceforge.net/
5. See http://jade.tilab.com/

working on the project, and had them take part in real test flights around the airfield. This allowed them to produce scenarios for the simulator. Participating in some of these real scenarios helped to make the scenarios concrete, thereby improving the software engineers' understanding of the domain, and ultimately the system.

Elicitation questions. The elicitation for the ATS system did not follow the questions in the order listed in Section 4.2, and we expect that this would be the case for other projects. The elicitation questions form a checklist, but the order in which they are asked did not seem important. Conversations were triggered by stakeholders, and we found it important to allow these conversations to occur and to record the details for further analysis, rather than fixating on the questions.

However, the questions did form the basis of some conversations, and our industry partners found this approach effective for bringing themselves into the agent mindset; something with which they had struggled previously.

Stakeholders as modellers. A key characteristic of our elicitation approach is the use of stakeholders as modellers. In our elicitation meetings, each stakeholder was provided with copies of the most recent models, and comments were invited. During the meetings, these models were modified by the group, thus taking advantage of the experts' knowledge of the domain. We found it necessary for the requirements team to perform further analysis in between meetings to ensure consistency between different models, etc., and to refine lower-level models.

We found that the lightweight nature of the models was useful in the meetings, as many incorrect assumptions that we had made were quickly identified by the domain experts, and corrected. This is consistent with our previous work with less-technical stakeholders.

Abstraction in understanding. The lower-level and less graphical models, such as the agent type specifications, were less useful in meetings, due to the inability to consider many of these models at one time. The motivational models (role models and goal models), were more useful, even at a high level. We found that lower-level models were largely produced outside of the round-table meetings. This is especially the case for the agent types. We believe that defining agent types outside of these meetings would be more straightforward than defining motivation models, due to having a better understanding of the system by this time.

The usefulness of high-level models is evident from an example. Early in the process, one of the requirements engineers devised a high-level domain model, which contained the concepts he thought were relevant, and links between these concepts. The links represented relationships, but did not define what these relationships were, as the engineer had not yet identified these. Initially, these were met with confusion from the other members in the meeting. The value of this model became clear only minutes later when one of the domain experts identified several incorrect assumptions about the relationships between concepts in the model, despite not knowing what the relationships meant. This indicates that, at least early in the requirements process, it is beneficial to share *any* understanding of the system, rather than waiting until models are complete.

Software system boundaries. We advocate delaying the definition of the system boundary until as late as is reasonable/possible, and at least until stakeholders have a shared understanding of the problem. Project or organisational constraints may require a system boundary early in the project, perhaps before the problem is fully understood. In these cases, we believe that the boundary decision should be delayed as long as possible without affecting the remainder of the project; e.g., contract agreements.

The software system boundary was left undefined for most of the requirements elicitation process. Our industry partners did not feel that it impacted the project negatively, however, in this particular case study, they did not see any benefit in delaying the boundary definition either, because they felt that the only obvious boundary was one in which all roles were played by software agents, although they did see that this could be useful for other systems.

To our group, the benefits of not defining a system boundary at the start of elicitation are illustrated by the project. We described earlier in this section the discussion that was held regarding whether the role of the *Manager* was to be played by a human or a software agent. Had the boundary been defined at the start of the requirements elicitation, this discussion may not have taken place. It is examples such as these that strengthen our claim that delaying definition of software system boundary can be beneficial.

The interaction designers we have collaborated with in other work [28], [31] have embraced the idea of delaying the software system boundary. A colleague (interaction designer) reported to us his experience with designing technology to support school children with autism. In working with a group of school teachers who were specialists in teaching autistic children, he found that by not constraining the system boundary, the teachers produced more useful solutions. By simply asking the teacher groups to design a technologically-based solution to support the children, the groups attempted to fit everything into software. The teacher groups who did not have this mandate all included technology as part of their designs, but the support was extended well outside of the software system boundary. This example captures the very essence of engineering socio-technical systems—that the people are as important as technology.

This is consistent with Gause's view [12], which states that taking the time in early requirements engineering to discuss possible solution boundaries with stakeholders raises awareness about possible solutions, and can discover *deep context regions*—those factors that are often oversights until a product is released.

Model evolution. As expected, our experience indicates that having models evolve over series of round-table discussions leads to a clearer solution, as early concerns regarding concepts such as resources were delayed without jumping to a pre-conceived solution. Later in the development process, successive versions of the models enabled traceability of the design decisions that were made, and of the requirements in general. This gave the research team something to fall back on when discussions started to get too complex for some stakeholders or drifted from original high-level goals, and also made the source of requirements more straightforward to trace. The example of the air traffic controller role

illustrates this, in which the models were updated to reflect this role, but even after its removal, parts of the model related to it remained. This is consistent with the findings described by MacLean et al. [19].

Model inconsistency. In our evaluation, reviews and cross-validation uncovered several major inconsistencies between agent types and other models (role and scenario models). This was the result of requirements engineers not explicitly considering the corresponding models in sufficient detail when defining the agent types. One possible solution to this is to modify the elicitation questions in Section 4.2.4 to draw agent type information from both role and scenario models. However, the resulting questions would be checks rather than elicitation mechanisms. A more suitable solution would be to incorporate these as checks (either manual or semi-automated) into the verification process, as the inter-model consistency could be checked by a tool. For example, Abushark et al. [1] present a method for inconsistency detection between agent-oriented models, which could be adapted for scenarios, roles, and agent types.

7.2 Requirements Specification and Packaging

Agent types. The major lesson that we learnt as part of this project was with regards to the inclusion of agent types in the SRS. In previous work, we had, like other researchers, considered agent types as a design artifact. Early in the ATS requirements process, our collaborators struggled to identify the behaviour of the system, or how they could implement and verify a system against a set of high-level models. This led us to the conclusion that the use of agent types to define behaviour is important; and therefore, the correctness of any implementation is the correspondence to the behaviour specified in the agent types. Once we added agent types to the SRS, the system behaviour became much clearer to our collaborators.

We believe that defining the behaviour is important with regards to obtaining a sign-off from the client. The signing off of requirements, and what constitutes this sign off, is overlooked in academic research on agent-oriented software engineering, but is important for contract definition in projects with third-party vendors.

Our collaborators at Jeppesen particularly like the flexibility enabled by the agent paradigm and the use of agent types. In their experience, clients on different projects are often happy to sign-off at different levels of abstraction. For example, some clients would be happy to sign-off the role and goal models, while others would want to see the more detailed agent types. In an event-based system, the distinction between different levels is less clear.

Repeatability. The evaluation of the three ATS projects demonstrates that our approach delivers requirements specifications that can be used to build systems in a repeatable manner. While the resulting designs contained differences between each other, all three designs and their corresponding prototypes were consistent with the SRS. Further to this, the developers had no previous experience in agent-oriented software engineering. This result was of particular importance to our industry partners, who aim to use agent-oriented models as tools for elicitation and communication both with internal developers, and with external stakeholders who are unlikely to have any background in agent-oriented modelling.

SRS Template. For the ATS project, the SRS of the system closely follows the structure recommended in this section, with some changes to suit the specific system. As part of the ATS specification, interaction models were derived, however, they were omitted from the latter versions of the SRS because we felt they did little to help define or understand the system behaviour. The other stakeholders agreed that the interaction models gave them little value in understanding the proposed solution. This is largely because the interaction protocols used in the ATS system were either largely sequential or were straightforward enough to extrapolate the interactions from the agent types and scenarios. The interaction diagrams were included in the software design.

In our view, the final SRS for the project is a well-packaged artifact. We believe the SRS to be correct, complete, and consistent, a view that is strengthened by our industry partners, who have endorsed (signed-off) the SRS package. This sign off is an agreement between ourselves and our industry partners that the requirements are correct, complete, and consistent, showing that our approach can be used to arrive at a solution with which all stakeholders are satisfied. We see this as an important result in itself.

8 Related Work

Agent-oriented requirements engineering has been investigated by other authors, and as a result, several methodologies have been proposed, such as Tropos [3], Prometheus [29], Gaia [44], INGENIAS [30], and ROADMAP [34]. Blanes et al. [2] performed a systematic overview of agent-oriented requirements engineering, finding that most research in the area focused on notations for modelling and analysis, with little support for requirements elicitation, specification (other than modelling), or validation.

8.1 Requirements Elicitation and Analysis

Both agent-oriented and goal-oriented requirements elicitation and specification have been investigated in the past. A key feature of much of the existing work is on motivations; that is, the "whys" of a system, in addition to the "whats". Our approach continues in this direction, and we have found this to be valuable in understanding systems.

Two major differences between our work and other agent-oriented and goal-oriented requirements elicitation methods are in the level of detail. First, similar to NFR [5], [24] and other works e.g [13], we acknowledge that committing to a design decision too early may result in some stakeholders' solutions being discarded, making their views irrelevant. However, a key contribution of this work compared with other work is that it further encourages stakeholder involvement by facilitating the inclusion of all key stakeholders in the *modelling* and *analysis* of the system, not just the elicitation. Zowghi and Coulin [46] note that, especially in group meetings, stakeholders must feel confident that their views will be heard, and that they are part of the process. We encourage stakeholders to discuss *and modify* models during group meetings help to engage them in the requirements process. In other work, we have successfully employed agent-oriented models in this context with

psychologists, general practitioners, sleep experts, nurses, ethnographers, and interaction designers [20], [25], [28], [31], [33]. This is facilitated by the fact that our approach relies on lightweight, hierarchical models. We believe that our approach could be modified to fit with other agent-oriented modelling notations that use concepts palatable to non-technical stakeholders, such as roles, goals, and agents.

Second, we prescribe a more detailed approach for eliciting information and recording it into models. Like us, Maiden et al. [21] identify that it is difficult to know where to start the modelling process. They recommend "kicking off" modelling with a context diagram before performing system modelling—a solution in the same spirit as our use of scenarios. We prefer to use scenarios rather than context diagrams because stakeholders often engage better with concrete examples, rather than abstract models.

The KAOS methodology includes a requirements acquisition technique [6] that is also based on questions. KAOS identifies what is required for the final models (system goals, agents, action, and domain attributes), but is less prescriptive in how to arrive at these. The Tropos methodology [3] uses a question-answer technique for eliciting requirements. The Tropos requirements elicitation technique involves four questions: *who are the main actors?*; *what are their goals?*; *how can they achieve them?*; and *does an actor depend on another to achieve its goals?*. These questions are broader versions of our questions for understanding the current system, and do not define how to arrive at solutions.

The social organisation metaphor has been used to analyse and specify requirements. Donzelli and Bresciani [8] use goal modelling to develop, during the analysis phase, an organisational view of agent-oriented systems. Yu [41] stresses the importance of identifying motivations within an organisational context in early-phase requirements engineering. Yu proposed the i^* modelling language to capture these motivations, commenting that social considerations are not commonplace among most modelling techniques. Yu's notion that software processes can be modelled as social processes is the essence of using agents to implement roles in our work. Maiden et al. [21] describe the application of i^* to the air traffic management domain, and present ten lessons learnt from this process.

Many of these concepts are inherited by the Tropos agent-oriented development methodology [3], which is built on i^*. Blanes et al. identify Tropos [3] as providing the most mature support for requirements elicitation. Like Tropos, our approach asks "why" as well as "what" when eliciting the requirements, because we agree that the motivation of the system is important for understanding how the system will fit within its organisation and environment. However, this research has not been translated into a standardised method to explicitly elicit requirements for agents using organisations as the guiding metaphor. Our approach provides a systematic and repeatable approach for eliciting requirements, which we believe could be used within the Tropos methodology.

8.2 Requirements Specification

A major difference between our work and other agent-oriented requirements engineering methods is the inclusion of agent types in the requirements specification. Typically, this is considered as design restriction, and therefore not good requirements engineering practice. However, the purpose of requirements engineering is to define the external behaviour of the system in its environment. Role specifications assign responsibilities for achieved goals, but purposely omit definitions of behaviour. Thus, multiple systems, each with different behaviour from the others, could achieve the specified goals.

We are not the first authors to identify that high-level conceptual models in agent methodologies are not sufficient to define behaviour. Ferber et al. [10] identify two approaches for specifying behaviour of a multi-agent system: assigning individual requirements to individual agents; and assigning behaviour to *role instances*, which are further refined into agents. The first specified behaviour from the perspective of an external observer, while the second specifies behaviour from the viewpoint of the individual instance, which is closer to our approach; however, we feel that the intermediate representation between roles and agents is unnecessary, and that our approach of assigning responsibilities to agents is a cleaner solution.

KAOS [6] defines the behaviour of systems using *agent/action* definitions. These are similar to our agent types, in that they define the agent and the actions that the agent can perform. KAOS does not distinguish between roles and agents, instead treating agents as the primary actors that achieve goals. The constraints related to goals are assigned to agents using *responsibility links*, making their notion of an agent a merging of our notation of roles and agents. When applying KAOS, van Lamsweerde et al. [37] comment that the last stages of the *goal elaboration* process "were performed in parallel with the agent/action identification and goal operationalisation". This provides further evidence that committing to some agent or activity design is necessary to define behaviour.

The Prometheus methodology [29], like KAOS, does not consider roles as part of requirements engineering. Like us, they identify that functionality must be considered to define behaviour. A Prometheus specification contains the system goals, but with no indication of the roles that achieve them. *Functionalities* are natural language descriptions of behaviour. Prometheus has been applied to industry applications, such as the meteorological alerting system developed with the Australian Bureau of Meteorology [23].

Maiden et al. [22] present a set of patterns for generating candidate requirements statements from i^* [41] models. Their goal is to bridge the gap from model-based specification to the common industry practice of requirements as lists of textual statements. The result is a list of requirements statements that can be traced to their corresponding i^* models. Applying this to an air-traffic control system, Maiden et al. demonstrated that the approach can reduce workload. In later work, Ncube et al. [27] automate this process, and an evaluation performed on the air-traffic control system requirements by four stakeholders showed that about two-thirds of these requirements were valid.

Ultimately, our approach, motivated by our industry partners, aims to move away from the practice of list of textual statements to using models themselves as

requirements. Our practice of using the agent types to define behaviour is one step towards this. One area of future work would be to derive a similar set of patterns to that used by Maiden et al. to generate templates for actions in agent types. This would help to alleviate the problem of inconsistency between models discussed in Section 7.1.

We have not seen other work that outlines how a requirements package should be constructed using agent-oriented models. Several other generic SRS package templates exist, such as that from Wiegers [38] and the IEEE Standard for requirements specifications [16]. These two templates both inspired our template, however, we see the value in using a template that emphasises the concepts that are central to the agent-oriented paradigm.

9 Conclusions

In this paper, we described two improvements to previous work on agent-oriented requirements engineering. These improvements relate to problems experienced by our industry partner: (1) a lack of systematic methods for agent-oriented requirements elicitation and modelling; and (2) a lack of prescribed deliverables for agent-oriented requirements.

Our elicitation approach prescribes a list of questions to be answered by stakeholders in round-table meetings, and how to directly map the answers to lightweight agent-oriented models. Further, we prescribe a requirements specification template that uses agent-oriented models as a central focus. Importantly, the template advocates the inclusion of agent types at the requirements level, rather than defining these at design time, based on our observation that roles, goals, and interactions alone are not sufficient for describing system behaviour.

We have been fortunate enough to attract an industry partner to work closely with to improve our requirements engineering processes—something that is difficult to do for researchers in this area. Applying our approach in conjunction with our industry partner demonstrated that the approach is useful, and we believe led to a much better requirements engineering method. A strong result of this work is that our industry partner has adopted many parts of our requirements engineering method into their own requirements engineering process models. This validates our claims that using the agent paradigm is not merely an academic exercise. Already our industry partner is applying their modified requirements engineering process model to the maintenance of large-scale air-traffic simulators. This domain is highly complex, so we expect to receive additional feedback.

In current work, we are applying our ideas in conjunction with other academic and industry partners, in a diverse range of domains including personal emergency alarms, depression care, and sleep disorders.

Acknowledgments

The authors would like to thank Alex Lopez for his valuable SRS reviews and cross validation; Daniel Thompson, Maksim Kozorez, and Mingwei Zhang for prototyping the ATS systems; and Keith Joshi, John Podlena, and Kayan Hau from Jeppesen for their highly valuable participation and discussions. This research is funded by the Australian Research Council Linkage Grant LP0882140.

References

[1] Y. Abushark, T. Miller, J. Thangarajah, and J. Harland, "Checking consistency of agent designs against interaction protocols for early-phase defect location," in *Proc. 13th Int. Conf. Auton. Agents Multi-Agent Syst.*, 2014, pp. 933–940.

[2] D. Blanes, E. Insfran, and S. Abrahão, "Requirements engineering in the development of multi-agent systems: A systematic review," in *Proc. Intell. Data Eng. Autom. Learn.*, 2009, pp. 510–517.

[3] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Auton. Agents Multi-Agent Syst.*, vol. 8, no. 3, pp. 203–236, 2004.

[4] B. Cheng and J. M. Atlee, "Research directions in requirements engineering," in *Proc. Int. Conf. Softw. Eng., Workshop Future Softw. Eng.*, 2007, pp. 285–303.

[5] L. Chung and J. Leite, "On non-functional requirements in software engineering," in *Conceptual Modeling: Foundations and Applications*, ser. LNCS, vol. 5600. New York, NY, USA: Springer, 2009, pp. 363–379.

[6] A. Dardenne, A. Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Sci. Comput. Programm.*, vol. 20, no. 1/2, pp. 3–50, 1993.

[7] S. DeLoach, M. Wood, and C. Sparkman, "Multiagent systems engineering," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 11, no. 3, pp. 231–258, 2001.

[8] P. Donzelli and P. Bresciani, "Improving requirements engineering by quality modelling—A quality-based requirements engineering framework," *J. Res. Practice Inf. Technol.*, vol. 36, no. 4, pp. 277–294, 2004.

[9] K. El Emam and A. Koru, "A replicated survey of IT software project failures," *IEEE Softw.*, vol. 25, no. 5, pp. 84–90, Sep./Oct. 2008.

[10] J. Ferber, O. Gutknecht, C. Jonker, J. Müller, and J. Treur, "Organization models and behavioural requirements specification for multi-agent systems," in *Proc. 10th Eur. Workshop Model. Auton. Agents Multi-Agent World*, 2001, pp. 1–19.

[11] D. Gause, "User driven design—The luxury that has become a necessity, a workshop in full life-cycle requirements management,"in *Proc. 4th IEEE Int. Conf. Requirements Eng. Tutorial T7*, 2000.

[12] D. C. Gause, "Why context matters-and what can we do about it?" *IEEE Softw.*, vol. 22, no. 5, pp. 13–15, Sep. 2005.

[13] V. Gervasi and D. Zowghi, "On the role of ambiguity in RE," in *Requirements Engineering: Foundation for Software Quality*, ser. LNCS, vol. 6182. New York, NY, USA: Springer, 2010, pp. 248–254.

[14] Standish Group, "Chaos report," (1994). [Online]. Avilable at: http://www.standishgroup.com

[15] R. Guizzardi and A. Perini, "Analyzing requirements of knowledge management systems with the support of agent organizations," *J. Brazilian Comput. Soc.*, vol. 11, no. 1, pp. 51–62, 2005.

[16] IEEE, *Recommended Practice for Software Requirements Specifications*, IEEE Std 830-1993, 1998.

[17] I. Jureta and S. Faulkner, "Clarifying goal models," in *Proc. ER (Tutorials, Posters, Panels & Industrial Contributions)*, 2007, vol. 83, pp. 139–144.

[18] A. Lopez, G. Beydoun, L. Sterling, and T. Miller, "An ontology-mediated validation process of software models," in *Proc. 19th Int. Conf. Inf. Syst. Develop.*, 2011, pp. 455–467.

[19] A. MacLean, V. Bellotti, and R. M. Young, "What rationale is there in design?" in *Proc. IFIP TC13 3rd Int. Conf. Human-Comput. Interaction*, 1990, pp. 207–212.

[20] M. Mahunnah, A. Koorts, and K. Taveter, "Towards distributed sociotechnical system for reporting critical laboratory results," in *Proc. Int. Conf. Health Inf.*, 2013, pp. 269–276.

[21] N. Maiden, S. Jones, C. Ncube, and J. Lockerbie, "Using i^* in requirements projects: Some experiences and lessons," in *Social Modelling for Requirements Engineering*, E. Yu, P. Giorgini, N. Maiden, and J. Mylopoulos, Eds. Cambridge, MA, USA: MIT Press, 2011, ch. 3, pp. 155–185.

[22] N. Maiden, S. Manning, S. Jones, and J. Greenwood, "Generating requirements from systems models using patterns: A case study," *Requirements Eng.*, vol. 10, no. 4, pp. 276–288, 2005.

[23] I. Mathieson, S. Dance, L. Padgham, M. Gorman, and M. Winikoff, "An open meteorological alerting system: Issues and solutions," in *Proc. 27th Australasian Conf. Comput. Sci.*, 2004, vol. 26, pp. 351–358.

[24] R. Mehta, H. Wang, and L. Chung, "Dealing with NFRs for smartphone applications: A goal-oriented approach," *Softw. Eng. Res., Manag. Appl.*, vol. 430, pp. 113–125, 2012.
[25] T. Miller, S. Pedell, A. Mendoza, L. Sterling, A. Kiernan, and A. Lopez-Lorca, "Emotionally-driven models for people-oriented software: The case study of emergency systems," (2014). [Online]. Avilable at: http://www.csse.unimelb.edu.au/~tmill/pubs/emotional-goals.pdf
[26] T. Miller, S. Pedell, L. Sterling, and B. Lu, "Engaging stakeholders with agent-oriented requirements modelling," in *Proc. 11th Int. Conf. Agent-Oriented Softw. Eng.*, 2011, pp. 62–78.
[27] C. Ncube, J. Lockerbie, and N. Maiden, "Automatically generating requirements from i* models: Experiences with a complex airport operations system," in *Proc. 13th Int. Working Conf. Requirements Eng.: Foundation Softw. Quality*, 2007, pp. 33–47.
[28] J. Paay, L. Sterling, F. Vetere, S. Howard, and A. Boettcher, "Engineering the social: The role of shared artifacts," *Int. J. Human-Comput. Studies*, vol. 67, no. 5, pp. 437–454, 2009.
[29] L. Padgham, and M. Winikoff, *Developing Intelligent Agent Systems: A practical guide*. New York, NY, USA: Wiley, Aug. 2004.
[30] J. Pavón and J. Gómez-Sanz, "Agent oriented software engineering with INGENIAS," in *Proc. 3rd Eastern Central Eur. Conf. Multi-Agent Syst. Appl.*, 2003, pp. 394–403.
[31] S. Pedell, T. Miller, F. Vetere, L. Sterling, S. Howard, and J. Paay, "Having fun at home: Interleaving fieldwork and goal models," in *Proc. 21st Annu. Conf. Australian Comput.-Human Interaction Special Interest Group*, 2009, pp. 309–312.
[32] A. Rao, "AgentSpeak(L): Bdi agents speak out in a logical computable language," in *Proc. 7th Eur. Workshop Model. Auton. Agents Multi-Agent World: Agents Breaking Away*, 1996, pp. 42–55.
[33] I. Shvartsman, K. Taveter, M. Parmak, and M. Meriste, "Agent-oriented modelling for simulation of complex environments," in *Proc. Int. Multiconf. Comput. Sci. Inf. Technol.*, 2010, pp. 209–216.
[34] L. Sterling and K. Taveter, *The Art of Agent-Oriented Modeling*. Cambridge, MA, USA: MIT Press, 2009.
[35] L. Sterling and K. Taveter, "Event-based optimization of air-to-air business processes," in *Proc. Intell. Event Process.-AAAI Spring Symp.*, 2009, pp. 80–85.
[36] A. Sutcliffe, "Scenario-based requirements engineering," in *Proc. 11th IEEE Int. Requirements Eng. Conf.*, 2003, pp. 320–329.
[37] A. Van Lamsweerde, R. Darimont, and P. Massonet, "Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt," in *Proc. 2nd IEEE Int. Symp. Requirements Eng.*, 1995, pp. 194–203.
[38] K. Wiegers, *Software Requirements*, 2nd ed. Bellevue, WA, USA: Microsoft Press, 2003.
[39] D. Wilmann and L. Sterling, "Guiding agent-oriented requirements elicitation: HOMER," in *Proc. 5th Int. Conf. Quality Softw.*, 2005, pp. 419–424.
[40] E. Yu, "Agent-oriented modelling: Software versus the world," in *Proc 2nd Int. Workshop Agent-oriented Softw. Eng. II*, 2002, pp. 206–225.
[41] E. Yu, "Social modeling and i^*," in *Conceptual Modeling: Foundations and Applications*. New York, NY, USA: Springer, 2009, pp. 99–121.
[42] E. Yu, P. Giorgini, N. Maiden, and J. Mylopoulos, *Social Modelling for Requirements Engineering*. Cambridge, MA, USA: MIT Press, 2011.
[43] F. Zambonelli, N. Jennings, and M. Wooldridge, "Organisational abstractions for the analysis and design of multi-agent systems," in *Proc. 5th Int. Workshop Agent-Oriented Softw. Eng.*, 2001, pp. 231–251.
[44] F. Zambonelli, N. R. Jennings, and M. Wooldridge, "Developing multiagent systems: The Gaia methodology," *ACM Trans. Softw. Eng. Methodol.*, vol. 12, no. 3, pp. 317–370, 2003.
[45] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 1, p. 30, 1997.
[46] D. Zowghi and C. Coulin, "Requirements elicitation: A survey of techniques, approaches, and tools," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds.. New York, NY, USA: Springer, 2005, ch. 2, pp. 19–46.