Experience from Building Industry Strength Agent-Based Appliances

Leon Sterling
The University of Melbourne
Department of Computer Science and Software Engineering
Victoria, 3010, Australia
leon@cs.mu.oz.au

Kuldar Taveter
The University of Melbourne
Department of Computer Science and Software Engineering
Victoria, 3010, Australia
kuldar@cs.mu.oz.au

The Daedalus Team
The University of Melbourne
Department of Computer Science and Software Engineering
Victoria, 3010, Australia
leon@cs.mu.oz.au

Abstract

This paper describes the Intelligent Lifestyle project conducted at the University of Melbourne in cooperation with our industrial partner Adacel Technologies in 2004. The Intelligent Lifecycle project aimed to design and build an industrial-strength system of intelligent agent-based appliances. It further aimed to demonstrate that agent development methods were suitable for wide acceptance by software developers with object-oriented but no agent-oriented experience. We first describe initial requirements engineering activities undertaken using the ROADMAP methodology and fast prototyping performed by using the RAP/AOR methodology. We next address the design and implementation of the intelligent agent system, focusing on a subsystem which handles communication in the case of an intruder break-in, and report on field testing. We finally analyse the results and draw conclusions. Overall, this paper defines and demonstrates a systematic approach for achieving systems of intelligent agents of industrial strength.

1. Introduction

A software agent is commonly understood as an *active* entity, possessing the features of *autonomy*, *proactiveness*, *responsiveness* and *social behaviour* [8]. Because of its distributed nature, a promising area of application for intelligent agents is in building a smart home where appliances interoperate seamlessly for the benefit of the home occupants. This application area can be naturally modelled as a society of interacting autonomous entities – agents.

The Intelligent Lifestyle project, conducted at the University of Melbourne in 2004, was part of an ongoing collaboration between Adacel Technologies and the Intelligent Agent Lab at the University of Melbourne, investigating how agent technologies can be effectively deployed in industrial and commercial settings. The collaboration includes an Industry Linkage Project to build a system of 'invisibly intelligent' appliances supported through the Australian Research Council, and a project to develop a methodology for developing agent-based systems for wide use supported by the Smart Internet Cooperative Research Centre. As part of this latter project, an Adacel engineer with no previous agent experience participated in application of the methodology. The Intelligent Lifestyle project aimed to design and build an industrial-strength system of intelligent agents, for the explicit purpose of performing field testing and providing demonstrations of intelligent agents. While the Intelligent Lifestyle project was initially accomplished by using the ROADMAP methodology [1, 6], we later complemented the software engineering process by fast prototyping using the RAP/AOR methodology [2].

In this paper, we first explain the requirements engineering activities conducted for the Intelligent Lifestyle project by means of the ROADMAP methodology [1, 6]. We then dwell on rapid prototyping by using the RAP/AOR methodology [2] and describe the simulation by using the JADE (JADE, http://jade.cselt.it/) agent platform [7]. After that, we address the design and implementation of a system of intelligent agents and report on the field testing that was performed at the University of Melbourne by a team consisting of 15 graduate and undergraduate students. Finally, we analyse the results and draw conclusions.

2. Requirements engineering

Initial requirements engineering for the Intelligent Lifestyle project was performed by means of the ROADMAP methodology [1, 6]. Figure 1 shows the models employed by the ROADMAP methodology. In the ROADMAP methodology, the models are divided vertically into Domain Specific Models, Application Specific Models and Reusable Services Models. The Environment Model and Knowledge Model represent

information about a specific domain and belong to multiple phases in the software development lifecycle. The Goal Model, Role Model, Agent Model and Interaction Model are tied to the system being modelled. Generic and reusable components in the system are captured by the Social Model and Service Model. The models are also split horizontally by dotted horizontal lines according to the analysis and design phases so that the Environment Model, Knowledge Model, Goal Model, Role Model and Social Model are parts of the domain analysis phase, while the Agent Model, Interaction Model and Service Model form parts of the design phase.

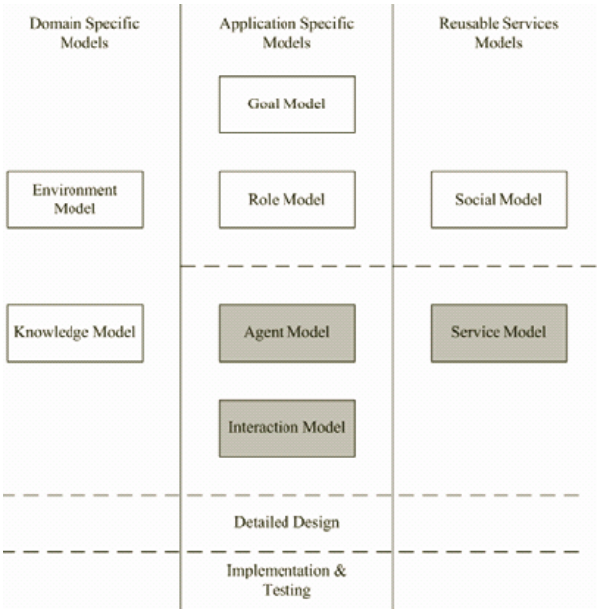

Figure 1. ROADMAP Analysis and Design Models.

2.1. The Environment Model and Knowledge Model

As the first step of requirements engineering, the Environment Model and Knowledge Model of the problem domain were created. The Environment Model consists of two high level environments – physical and conceptual – where each of them is further decomposed into specific zones. The conceptual environments are non-physical environments with which the system interacts, such as the Internet, Bluetooth and the SMS/GMS environments used in the Intelligent Lifestyle project.

The knowledge model is a way of formalizing the knowledge aspect of the system. It can be viewed as an *ontology* providing a common framework of knowledge for the agents of the problem domain. Figure 2 shows an example of a Knowledge Model component that was designed in order to enable the system to schedule activities and check users' schedules.

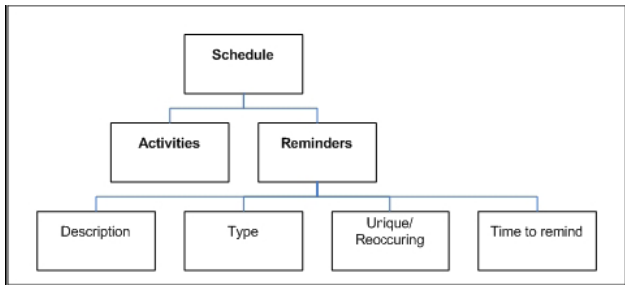

Figure 2. The Schedule Knowledge Component.

2.2. The Goal Models

The Goal Model provides a high level overview of the system requirements. Its main objective is to enable both domain experts and developers to pinpoint the *goals* of the system and thus the *roles* the system needs to fulfil in order to meet those goals. Implementation details are not described at all as they are of no concern in the analysis stage.

The Goal Model can be considered as a container of three components: Goals, Quality Goals and Roles. A Goal is a representation of a functional requirement of the system. A Quality Goal, as its name implies, is a non-functional or quality requirement of the system. We use the term *quality goal* instead of the artificial intelligence community's term of *soft goal* because we wish to emphasize the engineering aspects of the ROADMAP methodology that are necessary to ensure controlled delivery of desired outcomes. A *Role* is some capacity or position that the system requires in order to achieve its Goals. As Figure 3 reflects, Goals and Quality Goals can be further decomposed into smaller related sub-Goals and sub-Quality Goals.

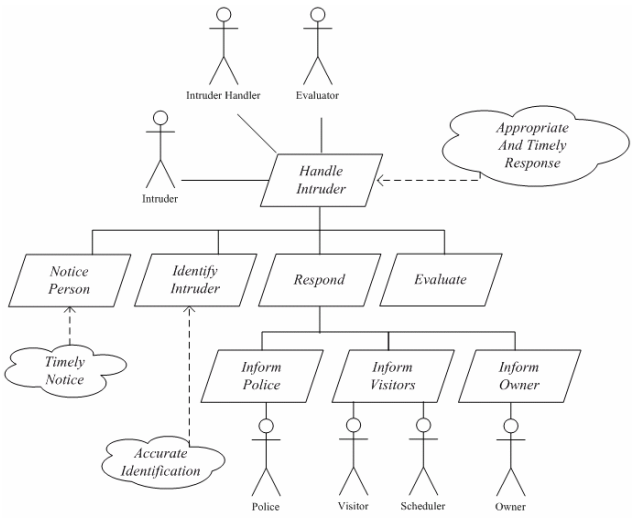

Figure 3. The Goal Model of intruder handling.

Figure 3 represents the Goal Model of the intruder handling scenario. In the diagram, the Role "Intruder Handler" has only one Goal which is to handle an intruder. This goal is characterized by the Quality Goal to provide an appropriate and timely response to a possible intruder detected. The Goal to handle an intruder can be decomposed into four sub-Goals – to notice person, identify intruder, respond and evaluate. There are the Quality Goals of timely notice and accurate identification pertaining to the sub-Goals to notice person and identify intruder, respectively. The sub-Goal to respond, in turn, has been divided into the sub-Goals to inform police, inform the visitors and inform the owner. To accomplish these, the additional Roles "Police", "Visitor", "Scheduler" and "Owner" are required. Please note that the order in which the sub-Goals are presented in Figure 3 does not *per se* imply any chronological order in which they are to be achieved.

2.3. The Role Models

The Role Model describes the properties of a Role. The term Role Schema is used interchangeably with Role Model. The Role Model consists of four elements to describe the Role:

Role Name: A name identifying the Role.
Description: A textual description of the Role.
Responsibilities: A list of duties/tasks that the Role must perform in order for a set of Goals and/or Quality Goals to be achieved.
Constraints: A list of conditions that the Role must take into consideration when performing its responsibilities. These include guard conditions, safety conditions and quality conditions.

Clearly, this is analogous to the delegation of work through the creation of positions in a human organisation. Every employee in the organisation holds a particular position in order to realise business functions. Different positions entail different degrees of autonomy, decision making and responsibilities. Taking this analogy, a Role Schema is the "position description" for a particular Role. Table 1 shows the Role Schema created for the Role "Intruder Handler" shown in the Goal Model in Figure 3.

The Role and Goal Models, as well as the Environment Model and Knowledge Model, were created in close cooperation with the client – our industry partner. This was facilitated by the use of the Roadmap Editor Built for Easy development (REBEL) tool [1] for building the Role and Goal Models.

Table 1. Role Schema for the Intruder Handler.

Role Name	Intruder Handler
Description	Identifies and responds to the intruder detected.
Responsibilities	Detect the presence of a person in the environment. Check the house schedule. Take an image of the person. Compare the image against the database of known people. Contact the police and send the image to them. Check the house schedule. Send a message to stay away to each visitor expected later that day. Inform the owner that the police are on their way and the visitors have been warned not to enter the house.
Constraints	The owner and each person pointed out by him/her needs to provide in advance personal information (face) to be recognised. A subject to be detected needs to be seen within the camera's image area. The schedule must be maintained and entered manually by the user. Visitors must be within the coverage area of mobile communication with their mobile access terminals switched on.

2.4. Fast Prototyping by RAP/AOR

Input and feedback from clients is critical in successful software engineering, so in order to enable our industry partner to provide feedback about the system to be developed at as early a stage as possible, the Environment Model, Knowledge Model and the Goal and Role Models were turned into executable specifications by using the RAP/AOR methodology. The Radical Agent-Oriented Process / Agent-Object-Relationship (RAP/AOR) methodology of software engineering and simulation has been introduced in [2] and is based on [3] and [4]. Before introducing executable models of intruder handling, we will explain briefly the notation to be used. For further explanations, please refer to [2].

An *external AOR diagram* specified by Figure 4 enables the representation in a single diagram of the types and instances of institutional, human and artificial (for example, software) agents[1] of a problem domain, together

[1] Agent type in RAP/AOR corresponds to Role in ROADMAP.

with their internal agent types and instances and their beliefs about instances of "private" and external ("shared" with other agents) object types. There may be attributes and/or predicates defined for an object type and relationships (associations) among agent and/or object types. A predicate, which is visualized as depicted in Figure 4, may take parameters.

Figure 4 shows that the graphical notation of AORML distinguishes between an *action event* (an event that is created through the action of an agent, such as a physical move performed by an Intruder) *type* and a *non-action event type* (for example, types of temporal events or events created by natural forces). The graphical notation of AORML further distinguishes between a *communicative* action event (or *message*) type and a *non-communicative* (physical) action event type like providing another agent with a commodity.

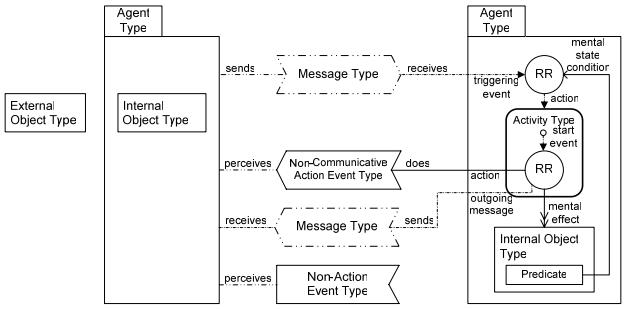

Figure 4. The core mental state structure and behaviour modelling elements of external AOR diagrams.

The most important behaviour modelling elements of AORML are *reaction rules*. As is shown in Figure 4, a reaction rule is visualized as a circle with incoming and outgoing arrows drawn within the rectangle of the agent type or instance whose reaction pattern it represents. Each reaction rule has exactly one incoming arrow with a solid arrowhead that specifies the *triggering event* type. In addition, there may be ordinary incoming arrows representing *mental state conditions* (referring to corresponding instances of other object types or to the predicates defined for them). There are two kinds of outgoing arrows for specifying the performance of *epistemic*, *physical* and *communicative actions*. An outgoing arrow with a double arrowhead denotes an epistemic action (changing beliefs). An outgoing connector to an action event type denotes the performance of a physical or communicative action of that type.

Reaction rules start activities. Each activity belongs to some *activity type* which we define as a prototypical job function in an organization that specifies a particular way of doing something by performing one or more elementary epistemic, physical and communicative actions in a non-zero span of time by an agent.

There are activity border events of the *start-of-activity* and *end-of-activity* types implicitly associated with the beginning and end of each activity. As Figure 4 reflects, the *start-of-activity* event type is graphically represented by an empty circle with the outgoing arrow to the symbol of the sub-activity type or internal reaction rule.

Figure 5 represents an external AOR diagram that models the scenario of intruder handling. The outermost activity of the scenario is started by reaction rule R1 that is triggered by an action event of the type move(IntruderDescription). This action event is created by an Intruder and perceived by the IntruderHandler. Note that the activity types modelled in the external AOR diagram in Figure 5 correspond to the Goals represented in the Goal Model in Figure 3. In other words, an activity of some type achieves a Goal of the respective type. For example, an activity of the type "Respond" achieves a Goal to respond to the detection of an Intruder. For the sake of clarity of Figure 5, the sub-activity type "Inform visitors", which involves the agent type Scheduler, is not refined in the figure. Reaction rule R2 represents checking of the Boolean value of the predicate isKnown attached to the object type Person. If the predicate evaluates to false, that is, if the person described by the IntruderDescription is not recognized, an activity of the type "Respond" is started.

Please note that the real scenario of intruder detection is more complicated than the scenario created for simulation purposes that is represented in Figure 5. For example, the real scenario includes checking the house schedule to see if someone, like a serviceman, has been scheduled to be in the house.

Reaction rules are also used for modelling elementary epistemic, physical and communicative actions. For example, reaction rule R1 in Figure 5 prescribes the creation of an instance of the shared object type IntruderDescription, while reaction rules R3 and R4 specify the performance of the respective communicative actions (sending of messages).

In [4] we have shown how external AOR diagrams can be straightforwardly transformed into the programming constructs of the Java Agent Development Environment (JADE, http://jade.cselt.it/) agent platform [7]. This enables fast prototyping by executable specifications. We turned the external AOR diagram represented in Figure 5 into the implementation constructs of JADE. As a result, we obtained a dynamic model which enables to simulate the scenario of intruder handling and experiment with it. In the model, the agent types IntruderHandler, Police, Owner and Scheduler were represented as the respective agent types of JADE. The shared object type IntruderDescription and the private object type Person were implemented as the corresponding Java object types and the predicate isKnown was represented in the form of a method attached to the object type Person.

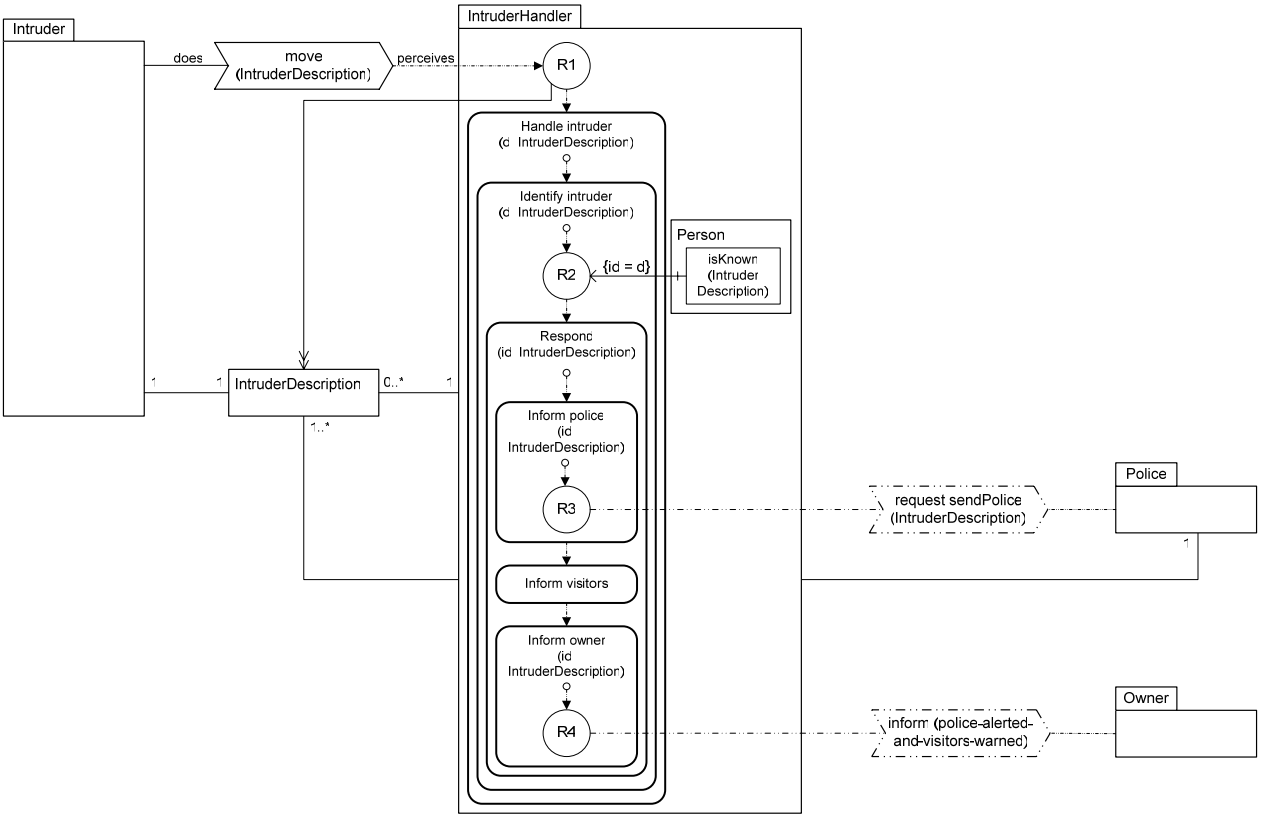

Figure 5. The external AOR diagram of intruder handling.

3. Design

Once the industry partner and the project team had a clear understanding of the system to be developed, it was appropriate to move on to the stage of designing an agent-based system. The design goals – the properties considered to be vital to the system – were:

- Adaptability in dynamic environment – Agents must be aware of context and adapt to changes. They should be able to change their plans during execution if the environment changes. This should be done as seamlessly to the human eyes as possible.
- Robustness – The architecture must facilitate consistent behaviour of the system as the system will be used by the industry partner for demonstration purposes. The system should always provide the required service whenever desired.
- Multi-agent collaboration – Agents must share information and work collaboratively to make correct deductions while achieving goals.

We will now address architectural and detailed design of the intruder handling system.

3.1. Architectural design

The project team identified two feasible ways to design a system of intelligent agent-based appliances. One way is to use a vertical layered architecture whereby information is passed up from low-level input devices, then to upper layers where that information is added to and converted to high-level representations finally ending up being used by applications. The other way is to use a horizontal design whereby agents provide information relating to their interests to other agents. In this way agents provide data as well as services to other agents and can thus access situational context from the variety of agents dealing with specific context. However, there is little control over contextual information which makes it hard to see how ideas such as conflicting information or information accuracy could be allowed for. History of information also seems difficult to add to such a design. In addition, the design cannot handle large (for example, more than a hundred) numbers of agents as it is complicated to produce a hierarchy of agents. Thus a large proportion of agents would be on the same level and use the same blackboard. This is not scalable. Considering our two options, therefore, the project team decided to adopt a vertical, two-tiered architecture consisting of an application tier and a context tier. The

Context Tier provides a broad base of information both directly gathered from input devices and interpreted through the analysis of various contextual information. The Application Tier comprises the usage of that information to provide services to users and interact with users through output devices, while the Context Tier does not have any access to output devices.

3.1.1. The Context Tier. There are three main ways to describe contextual information – centralized ("Blackboard"), decentralized ("Yellow pages") and hybrid (a combination of "Blackboard" and "Yellow pages").

The project team decided to use the "Blackboard" architecture because of its relative simplicity, allowance for conflict resolution and accuracy feedback. The Context Tier follows a layered architecture whereby entities within a layer are aware only of layers above and below them. Such a separation allows for a cleaner design and simplifies the later programming. It also satisfies the design goal of scalability as changes to a layer affect only the layer above it.

There are four layers in the architecture – sensors, cues, context soup and context resolution. *Sensors* represent sources of information. They are generally physical devices attached to a personal computer, such as a sensor for intruder detection, camera or microphone, although they could also be virtual sources of information such as the Internet. *Cues* have been introduced to decouple general information from specific and interpreted information. They extract specific information streams from sensors. For example, a microphone sensor may contain pitch, loudness, frequency as well as language information. Cues extract that information and, if needed, convert it into more accessible formats. The *context soup* consists of the blackboard that context providers post contextual information to along with the associated meta-context data, such as accuracy of the context data and information about the context provider. The blackboard is considered to be a black box in the architectural design. Context providers send information to the blackboard, where it is stored and its history is maintained. The *context resolution* layer comprises the interface to the Application Tier, as is shown in Figure 6. There are two main entities in this layer. The first is the Resolver which takes contextual information provided by context providers and complements or modifies the meta-context data. In the case of multiple sources of the same contextual information, the Resolver mediates between them to resolve conflicting data. The second entity in this layer is the Context Gateway which acts as a portal into the Context Tier for the Application Tier. Applications query the Context Gateway for contextual information that it retrieves from the blackboard and forwards to the applications.

3.1.2. The Application Tier. The Application Tier is the upper layer of the agent-based system of intelligent appliances where system-level services are provided. The Application Tier consists of two types of software agents. There are actuator agents that receive requests from other agents to perform an action using an actuator (that is, an output device). For example, actors of the types Police and Owner modelled in Figure 6 are represented by the CommunicatorAgent that contacts each actor in an appropriate way like through a Personal Digital Assistant (PDA) or a mobile phone. These agents are quite simple and will not be further described. There are also agents that are responsible for completing specific high level tasks, such as greeting a user, guiding a user from one location to another and handling an intruder. The overall design of these individual agents follows the Belief-Desire-Intention (BDI) architecture [5]. The beliefs of an agent in our system describe the situation the agent is in. The *beliefs* contain information the agent believes about its environment (that is, the contextual information). The beliefs also contain information that is internal to the agent. *Desires* are the situations that an agent is willing to bring about. They describe intentions the agent is supposed to achieve once certain beliefs about the environment or the agent itself hold. *Obligations* for an agent in the system are the services that the agent provides to other agents. They could be imagined as intentions that are imposed by external parties instead of internal beliefs. *Intentions* are actions to be taken to fulfil certain desires once certain beliefs are present. These actions may not be specific enough and are often too abstract to be executed by an agent. In order to carry out its intentions, an agent constructs plans, which are sequences of instructions or commands to be followed by it. Because plans consist of actions to be performed that will affect the external environment, they need to be resolved, that is checked against all other plans of the agent to make sure no conflict will result from trying to affect the environment.

Figure 6 details the architecture of the Application Tier in a graphical representation where the ovals represent processes or "processors" and the cylinders represent storages. As Figure 6 reflects, there are three layers in the Application Tier – Situation Assessment, Response Planning and Execution & Evaluation. The main purpose of the Situation Assessment layer is to produce the correct and complete set of beliefs. The three main objectives for the Response Planning layer are to maintain current desires and generate new desires, identify and decide to pursue the realizable intentions and produce a detailed step-by-step plan to carry out each intention to be realized. The Executor is responsible for interfacing with the physical actuators by generating and sending out correct commands to different devices, such as PDAs and mobile phones.

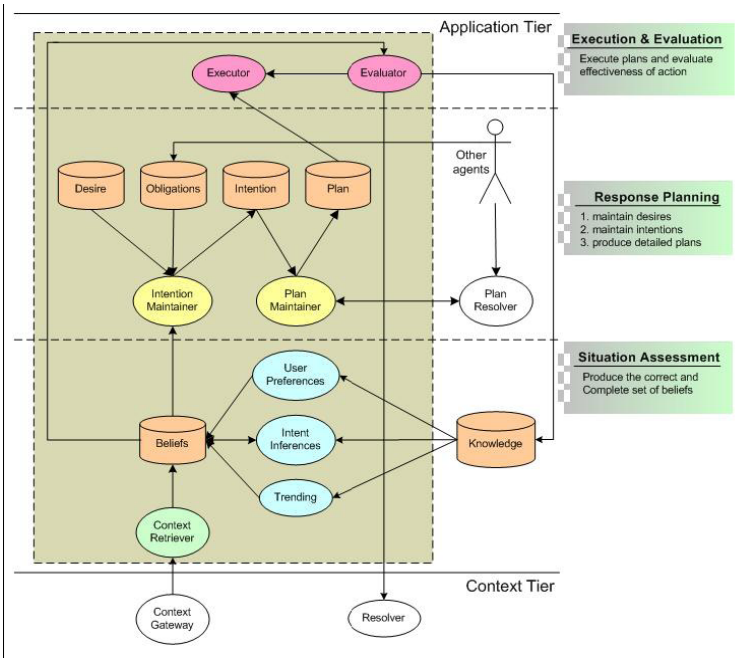

Figure 6. The Application Tier architecture.

3.2. Detailed design

For the scenarios to be implemented, including the scenario of intelligent intruder handling, the models of detailed design were produced.

Table 2. Comparison of agent frameworks.

Criterion	3APL	JADE	OAA
Familiarity with language and tools	3.5	4.0	4.5
Level of abstraction	4.5	3.5	3.0
Ease of deployment	2.0	4.0	4.0
Ease of debugging	2.5	3.5	4.0
Technical support	2.0	3.0	3.5
Documentation	2.5	4.5	4.5
Stability and maturity	2.0	3.0	3.5
How well does it fit into architectural design?	4.0	3.5	3.0
How many features need to be dropped from architectural design to use it?	4.5	4.5	4.5
The interface with other systems and code	3.5	4.0	3.5
Sum	**31**	**37.5**	**37.0**

The agents of the Application Tier need to communicate with each other. The project team decided that designing a communication mechanism from scratch would be beyond the scale of the project. For this reason, and because there are agent frameworks (infrastructures where inter-agent communication has already been defined and implemented) freely available, it was decided to pick an agent framework to be used. To this end, three freely available frameworks – JADE, Open Agent Architecture (OAA) and 3APL – were chosen for detailed evaluation.

Each framework was evaluated using the five-point scale according to the list of ten criteria as is reflected by Table 2. The values in the table have been achieved by averaging the points coming from each of the six members of the design team. As a result, the JADE agent platform was chosen due to its stability, language features and simplicity. This decision was made independently from using JADE for the fast prototyping described in Section 2.4.

Subsequently decisions of detailed design for the Context Tier and the Application Tier were made. For the Context Tier, the Web Server solution was chosen. Due to the necessity of having some context providers as non-Java programs, the extra flexibility of having a Web Server accessible from any language outweighed the tedium of coding the Web Server.

In the Web Server design of the Context Tier depicted in Figure 7, the context providers post information to the Web Server which resolves any conflicts between contextual information, thus acting

as the Resolver described in Section 3.1.1. The Web Server then requests the Blackboard Manager to add the contextual information to the storage of context data. In this way, the Context Gateway has direct access to resolved contextual information, the Resolver (Web Server) is responsible for inserting contextual information and the Blackboard Manager deletes and updates contextual information as needed.

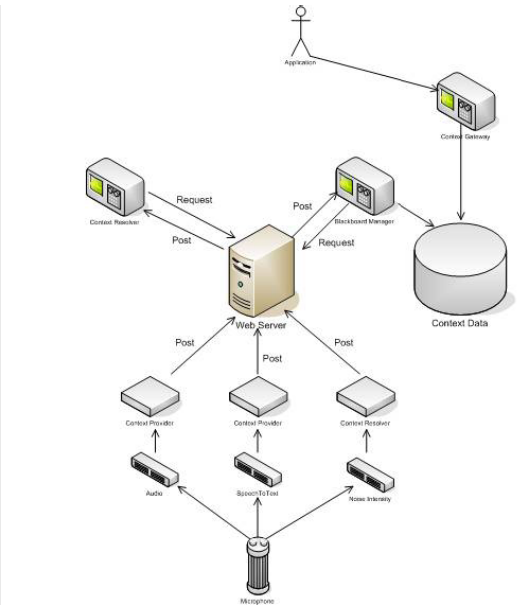

Figure 7. The Web Server design of the Context Tier.

Figure 8 shows the class diagram depicting detailed design of the Application Tier. As it can be seen from the diagram, there are three agent classes – BDIAgent, PlanResolver and TextToSpeech – that extend the Agent class implemented by the JADE framework. The BDIAgent is the base agent class that gets extended by more specific agent classes of the Application Tier, such as Greeter represented in Figure 8. It has abstract methods that are implemented by agent classes extending this class. An instance of BDIAgent has reference to one or more Context objects (not shown in Figure 8), one or more Intention objects and exactly one Plan object. A Context object is used to encapsulate contextual information. The Context Gateway of the Context Tier will send an object of this class to the agents interested in and registered for receiving contextual information of the corresponding type. The Intention class is an abstract class that describes Intention objects. An Intention object has the attribute that is used to specify one of two types: obligation or intention. An obligation is a request from another agent while an intention results from a combination of the agent's beliefs. The Intention class can be extended for different functionalities provided by the Application Tier. For example, Figure 8 shows that a greeting intention is represented by a GreetIntention object that has attributes greetee and timeToGreet so that the intention of who and when to greet can be specified clearly by the object. The Plan object represents a plan formulated by a BDIagent. A plan has an intention corresponding to this plan, a priority level and a vector of actuators (output devices) to be used.

The Greeter class was the only agent class extending the BDIAgent class in the first build of the system of intelligent agent-based appliances. It was accompanied by the Guider and IntruderHandler classes in the second and third builds.

The second class extending the Agent class implemented by the JADE framework is PlanResolver. Its instance receives a plan sent by a BDIAgent and either approves or disapproves it.

The third class represented in Figure 8 that extends the Agent class of the JADE framework is Text-ToSpeech. Its instance receives from a BDIAgent a message to be executed and calls a module implemented in C++ that converts text to speech.

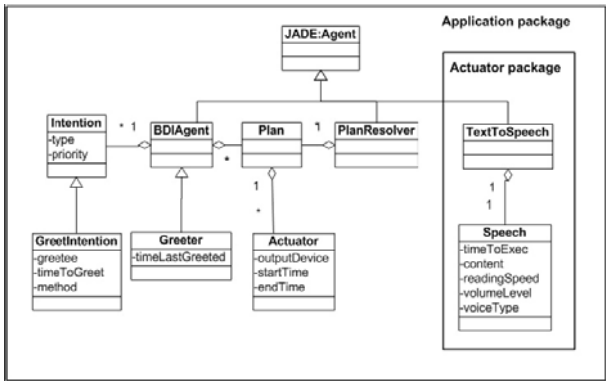

Figure 8. The class diagram of the Application Tier.

4. Implementation and testing

In addition to the design decisions reported in Sections 3.1. and 3.2., the implementation was based on some decisions that depended on the available technology. For example, it appeared that a human agent playing the Owner role was needed to decide whether a person detected should be treated as an intruder because the technology required for comparing an image of a person against the database of known people was not mature enough.

We performed field testing with the intruder handling scenario. In the scenario, when a person enters the house area, the Image Recogniser (Web camera) detects the presence of a person and captures

his/her face. Then, the Context Gateway sends the contextual information pertaining to the person detected from the Context Tier to the IntruderHandlerAgent of the Application Tier. This agent is intended to run in parallel with humans acting out the scenario. It first calls the SchedulerAgent to check the house schedule to see if someone, for example, a serviceman has been scheduled to be in the house at that time. After that, the agent checks with the owner whether he/she knows the person detected. Once the owner has answered "No" (see Figure 9) and no one has been scheduled to be in the house, the IntruderHandlerAgent contacts the police to report the intrusion. Followingly, the IntruderHandlerAgent again calls the SchedulerAgent to check the house schedule to see whether there are any visitors scheduled to come during a specified time period. If there are scheduled visitors, the SchedulerAgent retrieves a list of their names and contact data and sends a message with this information to the IntruderHandlerAgent. Subsequently, the IntruderHandlerAgent contacts the scheduled visitors to warn them to stay away. Finally, the IntruderHandlerAgent informs the owner that both the police and visitors have been contacted. Messages with the owner, police and visitors are exchanged through the CommunicatorAgent.

In the course of field testing performed during the project, the project team found some technologies applied, such as JADE, PDA and Microsoft Speech (for text-to-speech and speech-to-text conversions), to be very useful for producing solutions of industrial strength. However, some other technical solutions applied, such as OpenCV used for image processing and Bluetooth, did not perform satisfactorily. The project team also recognized the opportunities for using alternative technologies, like mobile phones and SMS-messages instead of PDAs.

In the course of testing, heuristic evaluation of the system was performed by three student users who were different from implementers. The following metric was used for the scoring system of heuristic evaluation:

- Level 1: the application fulfils this particular criterion 0 – 20% of the time.
- Level 2: the application fulfils this particular criterion 21 – 40% of the time.
- Level 3: the application fulfils this particular criterion 41 – 60% of the time.
- Level 4: the application fulfils this particular criterion 61 – 80% of the time.
- Level 5: the application fulfils this particular criterion 81 – 100% of the time.

Table 3. Heuristic evaluation of the Intelligent Lifestyle application.

Criterion	User 1 (PDA)	User 2 (STT)	User 3 (TTS)
Response time	0	N/A	4
Visibility of messages by the system	4	N/A	N/A
Understandability of messages by the system	3	N/A	3
Error prevention	2.5	N/A	N/A
Reversible actions	0	N/A	N/A
Feedback by the system	1	3	4
Consistency	4	5	N/A
Normal interaction with voice recognition	N/A	5	N/A
Accuracy of voice recognition	N/A	5	N/A
Friendliness of the system's response	N/A	N/A	3
Variety of phrases and tones used by the system	N/A	N/A	1
Human-like tone	N/A	N/A	2
Audibility of system's response	N/A	N/A	2
Average	**2.07**	**4.5**	**2.71**

Different users focused on testing the following features of the system: Personal Digital Assistant (PDA), Speech-To-Text (STT) and Text-To-Speech (TTS) conversions. Therefore not all criteria were considered by each user. The feedback given by the users is summarized in Table 3. The testing results presented in Table 3 reflect that the usability of the application could be further improved.

The video made of the field testing is available as http://www.cs.mu.oz.au/~kuldar/final.swf

Figure 9. A snapshot of the video made of the field testing.

5. Conclusions

The models produced by our requirements engineering activities proved useful for understanding the problem domain. Moreover, it was possible to execute (simulate) domain models which helped to understand how the system to be designed should look and function. However, we found that a real industrial-strength implementation required architectural and design decisions that could not be predicted from the domain analysis phase. In other words, it appears to be hard to achieve the design and implementation of a robust, efficient software system by just extending the models of the problem domain. This seems to be due to software Quality Goals falling into two separate categories: qualities desired of the functionality of the system, such as timeliness, accuracy and politeness, and qualities desired of the construction of the system, such as robustness, efficiency and scalability. These categories are clearly related, but the relation is neither simple nor obvious, and requires further work by both the software agents' and the software engineering communities. For example, there was no straightforward manner in which the external AOR diagram represented in Figure 5 could have been turned into the design model and implementation consisting of the Context and Application Tiers. We believe that this was at least partly due to the fact that some design decisions were influenced by trends in the software agents' community. The fast prototyping via RAP/AOR reported in Section 2.4. made it clear that the scenario of intruder handling does not really need complicated BDI agents but could be successfully implemented by means of less sophisticated reactive agents instead. Based on the experience of transforming external AOR diagrams into the programming constructs of JADE reported in Section 2.4., it should be possible to generate, from domain models, design and implementation solutions consisting of reactive agents. As reflected in Figure 1, this would be consistent with the ROADMAP methodology, which covers the whole process of software engineering from requirements to implementation. Automated software engineering of this kind would also comply with the Model Driven Architecture (MDA, http://www.omg.arg/mda/) where computation-independent domain models are transformed into platform-independent design models that are then turned into platform-specific models and implementations. We intend to investigate automated software engineering from domain models to reactive agents in the future.

We also believe that different tools are necessary for achieving automated agent-oriented software engineering. This project served as a useful test case for developing the Roadmap Editor Built for Easy development (REBEL) tool [1] for building Goal Models and Role Models.

We also found that in the future, more emphasis should be put on exception handling. For example, how to proceed if the owner is not contactable?

6. Acknowledgements

We are thankful to the Australian Research Council (Industry Linkage Project number 0348797), the Smart Internet Cooperative Research Centre (grant number SPA-07), the Adacel Technologies and to Kendall Lister and Thomas Juan.

7. References

[1] Kuan, P. P., Karunasakera, S., Sterling, L. Improving Goal and Role Oriented Analysis for Agent Based Systems. In *Proceedings of the 16th Australian Software Engineering Conference* (ASWEC 2005), 31 March - 1 April 2005, Brisbane, Australia. IEEE 2005, 40-47.

[2] Taveter, K., Wagner, G. Towards radical agent-oriented software engineering processes based on AOR modelling. In B. Henderson-Sellers & P. Giorgini (Eds.), *Agent-oriented methodologies* (pp. 277-316). Hershey, PA: Idea Group, 2005.

[3] Wagner, G. The agent-object-relationship meta-model: Towards a unified view of state and behavior. *Information Systems, 28*(5), 2003, 475-504.

[4] Taveter, K. *A multi-perspective methodology for agent-oriented business modelling and simulation*. PhD thesis, Tallinn University of Technology, Estonia, 2004.

[5] Rao, A. S., Georgeff, M. P. BDI Agents: From Theory to Practice. In Victor R. Lesser and Les Gasser (Eds.), *Proceedings of the First International Conference on Multiagent Systems*, June 12-14, 1995, San Francisco, California, USA (pp. 312-319). The MIT Press, 1995.

[6] Juan, T., Sterling, L. The ROADMAP Meta-model for Intelligent Adaptive Multi-agent Systems in Open Environments. In P. Giorgini, J. P. Muller, J. Odell (Eds.), *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers* (LNCS, Vol. 2935, pp. 826-837). Berlin: Springer-Verlag, 2003.

[7] Bellifemine, F., Poggi, A. & Rimassa, G. (2001). Developing multi-agent systems with a FIPA-compliant agent framework. *Software – Practice and Experience, 31* (2001), 103-128.

[8] Jennings, N. R., Sycara, K., Wooldridge, M. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems, 1*(1), 7-38, 1998.