Sterling, L., Taveter, K., & The Daedalus Team. (2006). Building agent-based appliances with complementary methodologies.


Copyright © 2006 The authors.
This is the author’s version of the work, posted here with the permission of the publisher for your personal use. No further distribution is permitted. You may also be able to access the published version from your library. The definitive version is available at http://www.iospress.nl/.
Building Agent-Based Appliances with Complementary Methodologies

Leon Sterling a, Kuldar Taveter a, 1 and The Daedalus Team a

a The University of Melbourne, Department of Computer Science and Software Engineering, Australia

Abstract. This paper describes a part of 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 a 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. In this paper we describe how initial requirements engineering activities undertaken using the ROADMAP methodology can be complemented by rapid prototyping performed by using the RAP/AOR methodology. Overall, this paper defines and explains the first two stages of a systematic approach for achieving systems of intelligent agents.

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 a system of intelligent agents, for the explicit purpose of performing field testing and providing demonstrations of intelligent agents.

Since no specific agent-oriented methodology supports all possible quality goals of a software system to be created, such as politeness and robustness, it is feasible to use a combination of two or more methodologies as has been proposed in [10]. In the Intelligent Lifestyle project, the ROADMAP methodology [1, 6] was used for

1 Corresponding Author: The University of Melbourne, Department of Computer Science and Software Engineering, Victoria, 3010, Australia; E-mail: kuldar@cs.mu.oz.au
requirements engineering and the RAP/AOR methodology [2] – for rapid prototyping. Additionally, object-oriented methods were used for design and implementation.

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]. Finally, we analyse the results and draw conclusions. Since the design and implementation of the system of intelligent agents worked out in the Intelligent Lifestyle project and its field testing have been described in the report [9] that is available online, we do not address them in this paper. In addition to this, the video made of the field testing is available as http://www.cs.mu.oz.au/~kuldar/final.swf (select Scenario and Intruder Scenario).

1. Customizing agent-oriented methodologies by reusing features

In [10] is proposed a modular approach enabling developers to build customized project-specific methodologies from agent-oriented software engineering (AOSE) features. An AOSE feature is defined in [11] to encapsulate software engineering techniques, models, supporting Computer-Aided Software Engineering (CASE) tools and development knowledge such as design patterns. It is considered a stand-alone unit to perform part of a development phase, such as analysis or prototyping, while achieving a quality attribute such as privacy. Another method for building customized methodologies – OPEN [12] – includes the notions of Work Units, Work Products, Producers, Stages and Languages. We can define an AOSE feature in terms of these notions as a Work Unit performed by one or more Producers in support of a specific software engineering Stage resulting in one or more Work Products represented in the respective Languages. For example, we can identify the feature of goal and role modelling performed by a domain engineer in support of the requirements engineering stage and resulting in Goal and Role Diagrams and Role Schemas. Analogously, we can identify the feature of simulation performed by a domain engineer and system architect in support of the rapid prototyping stage and resulting in the executable constructs of the JADE agent platform represented in the Java language. Both of the features mentioned support the Quality Goals of adequacy and correctness of the requirements captured.

According to [12], a work product is any significant thing of value (e.g., document, diagram, model, class, or application) that is developed during a project. A language is the medium used to document a work product. A producer is anything that produces (i.e., creates, evaluates, iterates, or maintains), either directly or indirectly, versions of one or more work products. A work unit is defined as a functionally cohesive operation that is performed by a producer. A stage is a formally identified and managed duration of time. Table 1 presents the Stages, Work Units, Producers, Work Products and Languages of the Intelligent Lifestyle project. As we implied in the paragraph above, features are orthogonal to these notions of OPEN. Table 1 thus shows how AOSE features originating in different methodologies complement each other. However, differently from OPEN [12], we do not regard it as necessary to rely on the formal metamodel of features. As has been demonstrated by our earlier work [6, 10, 11], informal approach to methodology composition works equally well and is more likely to be adopted by industry.
Table 1. Stages, Work Units, Producers, Work Products and Languages of the Intelligent Lifestyle project.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Work Units</th>
<th>Producer(s)</th>
<th>Work Product(s)</th>
<th>Language(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements engineering</td>
<td>Goal and role modelling, environment modelling, knowledge modelling</td>
<td>Domain engineer</td>
<td>Goal and Role Diagram, Role Hierarchy Diagram, Role Schema, Environment Model, Knowledge Model, System Overview Diagram</td>
<td>ROADMAP Graphical Modelling Language</td>
</tr>
<tr>
<td>Rapid prototyping</td>
<td>Process modelling, simulation</td>
<td>Domain engineer, system architect</td>
<td>External AOR Diagram, constructs of JADE agent platform</td>
<td>AOR Modelling Language, Java</td>
</tr>
<tr>
<td>Architectural design</td>
<td>Logical architecture modelling, hardware architecture modelling</td>
<td>System architect</td>
<td>Software Architecture Diagram, Hardware Architecture Diagram, UML Interaction Diagram</td>
<td>Based on UML or UML</td>
</tr>
<tr>
<td>Detailed design</td>
<td>Designing individual components (agents) of the system</td>
<td>Software engineer</td>
<td>Component Architecture Diagram, UML Class Diagram, UML Interaction Diagram, UML State Machine Diagram</td>
<td>Based on UML or UML</td>
</tr>
<tr>
<td>Implementation</td>
<td>Implementing agents</td>
<td>Software engineer, Deployment engineer</td>
<td>UML Deployment Diagram, constructs of JADE agent platform</td>
<td>Java</td>
</tr>
</tbody>
</table>

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.
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.
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.

![Diagram](image)

**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.

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Intruder Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Identifies and responds to the intruder detected.</td>
</tr>
<tr>
<td><strong>Responsibilities</strong></td>
<td>Detect the presence of a person in the environment. Check the house schedule for strangers scheduled to be there. 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 for planned visitors. Send a message to stay away to each visitor expected that day. Inform the owner that the police are on their way and the visitors have been warned not to enter the house.</td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td>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 user must maintain the schedule. Visitors must be within the coverage area of mobile communication with their mobile access terminals switched on.</td>
</tr>
</tbody>
</table>

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 2 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 customer – 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.

3. Rapid prototyping by RAP/AOR

Input and feedback from customers 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 an early 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 (that is, modelled from the perspective of an external observer) 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 of a problem domain, together 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 RAP/AOR 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 RAP/AOR 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.

The most important behaviour modelling elements of RAP/AOR 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

---

2 Agent type in RAP/AOR corresponds to Role in ROADMAP.
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.

<table>
<thead>
<tr>
<th>Agent Type</th>
<th>Message Type</th>
<th>Non-Action Event Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Type</td>
<td>Event Type</td>
<td>Event Type</td>
</tr>
<tr>
<td>Non-Action</td>
<td>Event Type</td>
<td>Event Type</td>
</tr>
<tr>
<td>Event Type</td>
<td>Event Type</td>
<td>Event Type</td>
</tr>
<tr>
<td>Event Type</td>
<td>Event Type</td>
<td>Event Type</td>
</tr>
<tr>
<td>Event Type</td>
<td>Event Type</td>
<td>Event Type</td>
</tr>
</tbody>
</table>

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

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. The precise mechanism of perceiving is of no interest at this stage of a software engineering process. 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 anyone, like a serviceman, has been scheduled to be in the house.

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 rapid
prototyping by executable specifications. As a result of turning the external AOR diagram represented in Figure 5 into the implementation constructs of JADE, 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 software 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.](image)

4. Conclusions and related work

The models produced at our requirements engineering and rapid prototyping stages proved useful for understanding the problem domain. Moreover, it was possible to simulate domain models which helped to understand how the system to be designed should look and function. However, at later stages we found that a real implementation required design decisions that could not be predicted from the requirements engineering 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. To tackle the problem, we plan to complement the ROADMAP methodology by other types of diagrams enabling to associate quality attributes of the latter type with specific software architectures and platforms. Also, we are currently working on adding features suitable for design from the Prometheus [13] methodology.
The Intelligent Lifestyle project demonstrated the value of rapid prototyping which enables receiving useful feedback from customers at an early stage of a software engineering process. This helps to avoid costly mistaken design decisions.

We believe that different tools are necessary for achieving automated agent-oriented software engineering, including rapid prototyping. 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.

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, Clarinox Pty Ltd and to the members of the Agentlab of UoM.

References