

Copyright © 2004 IEEE.

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://ieeexplore.ieee.org/.
Adapting Roles for Agent-Oriented Software Engineering

Kevin Chan, Leon Sterling
Department of Computer Science and Software Engineering
University of Melbourne, Australia.
{kchan, leon}@cs.mu.OZ.AU

Abstract

Roles have a limited use within object-oriented (OO) analysis and design. In extending OO concepts to agent-oriented software engineering (AOSE), roles must be adapted to incorporate autonomy and other agent essentials. In this paper, we present our perspective on roles and how to specify them. We also introduce an extension to roles for design, namely agent classes. We describe how roles and agent classes can be used in practice during the requirements and architectural phases of AOSE.

1. Introduction

People have long been designing complex human systems (e.g. universities and companies) in terms of interacting, intelligent human agents playing a number of different roles. Using roles to describe software systems may enable software engineers to leverage off this familiarity in order to conceptualize and describe complex software system with greater ease.

In Object-Oriented (OO) design, there are classes, interfaces, and roles. A class can realize a number of interfaces, and roles are used to specify which of its interfaces an instantiated object presents to the world in a particular context [2]. The role that an object plays determines the behavior that the object’s clients can expect of it. These concepts enable the interactions between objects to be more clearly specified, and are also critical for specifying open, extensible systems. This paper is concerned with adapting these concepts for use in agent-oriented (AO) design.

As software agents become an increasingly active area of research, agent-oriented software engineering (AOSE) has emerged as a significant sub-area. AOSE is about defining methods for creating predictable, reliable, and stable agent-based systems. AOSE methodologies that have been proposed in recent times include Tropos [3], Gaia [13], MESSAGE [15], Prometheus [10], and ROADMAP [6].

Most current AOSE methodologies [13] [6] [10] use a concept of “role” during the requirements development phase. While we agree that roles are useful for analyzing requirements, roles (as defined in these methodologies) contain too much implementation bias, mixing design decisions into the requirements phase.

In this paper, we describe “agent roles”, a new construct for specifying roles that avoids implementation bias. We also describe an extension to agent roles called "agent classes" that include some design information. These constructs can be used to describe the requirements and architectural design of agent-based systems.

Agent roles and agent classes can be thought of as AO adaptations of OO interfaces and OO classes, respectively. As a result, specifying AO systems with agent roles and agent classes has a similar flavor to specifying OO systems with interfaces and classes. Because OO is the current industry standard for designing software systems, this similarity may make it easier for current industry practitioners to move into AO development. To further support this, agent role and agent class specifications have a level of detail and formalism that is similar to UML, the de facto standard for modeling object-oriented systems.

[Overview of paper structure]

2. Agent Roles, Agent Classes, and Agents

To specify agent-based systems, we employ three concepts: agent roles, agent classes, and agents (see Figure 1). In this section, we provide an overview of these concepts, and describe how these concepts relate to each other.
Agent Role:
Agent roles are at the requirements level. Agent role specifications describe what a role will do without specifying how the role will do it. When analyzing a system, we begin by representing the requirements as a set of agent roles. Agent roles are analogous to OO interfaces.

Agent Class:
Agent classes are at the architectural design level. While agent roles are free of implementation bias, agent classes begin to capture design decisions and trade-offs. They are analogous to OO classes.

An agent class specifies one particular way of fulfilling a role. There can be more than one agent class for a given role, as there can be multiple ways of fulfilling a role. It is also possible that a single agent class simultaneously fulfills a number of different roles.

Agent classes may use other roles when fulfilling the given role. Agent classes use the functionality of other agents via their roles. An agent class may use as few or as many other roles as it requires. A role may be used by any number of agent classes.

The relationship between an agent class and the roles that it uses forms a hierarchical structure. The roles used by an agent class are fulfilled by agent classes which may in turn use other roles. This hierarchy of roles and agent classes enables an agent-based design to be created using structured, iterative refinement.

Agent:
Agents implement the design described by an agent class. Depending on which agent classes it implements, an agent can play a number of roles.

The roles that an agent plays determine how other agents can interact with it. Agents interact with each other via their roles. An agent can implement a number of roles, and a role can be implemented by a number of agents.

Agents implement agent classes in order to play roles. An agent class specifies one particular way of fulfilling a role. An agent can implement multiple agent classes, and agent classes can be implemented by multiple agents.

An agent may implement a set of agent classes that fulfills a number of different roles. This would enable the agent to play a number of different roles. An agent may also implement multiple agent classes that fulfill the same role. This would enable the agent to play the one role in a number of different ways. This allows the agent to flexibly choose the most appropriate way to play the role for each situation. An agent may also mix and match, playing multiple roles and playing some roles in multiple ways.

3. Agent Roles

As mentioned earlier, agent roles are a requirements level concept. In this section, we describe “agent role specifications”, our method for specifying agent roles. Agent role specifications are focused on what an agent role does, and do not describe how the functionality should be achieved. (How agent roles are fulfilled is a design decision, and is left to the agent class specifications.)

We specify agent roles in terms of:
- **Responsibilities**: What the agent playing this agent role will do in reaction to changes in the environment. Many of an agent role's functionality can be specified as responsibilities.
- **Initiatives**: Proactive behaviors, tasks that the agent will perform even if it's not reacting to any specific event.
- **Facilities**: Tasks that other agents can request of the agent playing this agent role. These are like operations in OO, except that agents have autonomy meaning that the agent can decide whether it will perform the request.

We call the Responsibilities, Initiatives and Facilities that comprise an agent role specification the "role specification elements" of an agent role.

Agent role specifications should be as cohesive as possible. The role specification elements of an agent role should be targeted at a single purpose. If it makes sense for an agent to have a number of related but different purposes, the agent should play a set of agent roles. We should not create bloated agent roles for these cases.


**Responsibilities**

Responsibilities specify the reactive behavior of an agent role. Responsibilities are specified as follows:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Page Added</td>
<td>Index is updated to include new page within 24 hours</td>
</tr>
</tbody>
</table>

This describes the condition for activating the responsibility, the outcome that the agent should achieve when this responsibility is activated, and any constraints on this responsibility.

For example, suppose we had an Indexer agent role that was responsible for maintaining the index for a web site's internal search engine. One of its responsibilities may be:

New Page Added → Index is updated to include new page

Within 24 hours

This specifies that when a new page is added, the index should be updated to include the new page within 24 hours after the page is added.

We do not specify how the "Condition" is detected. We only state what the condition is. We want to keep this flexible at the agent role specification level. An agent may be able to detect this in multiple ways, and we do not want to constrain the way it does it yet. For example, the agent may be able to detect a new page by waiting for an event from another agent. Or, if that's not available, the agent may have to poll/monitor the directories where web files might be added. New methods may appear as new agents (playing new agent roles) are introduced into the system. The agent role specification should be flexible enough to support future enhanced implementations that cannot be implemented currently. We leave the details of how it is done to the agent class specification.

We specify the "Outcome" required in response to the "Condition" rather than the tasks that the agent should perform. Again, we want to keep this flexible in the agent role specification. An agent may achieve the desired outcome in a number of different ways; we do not want to constrain how an agent achieves the required outcome. Using the example from before, an agent implementing the Indexer agent role may choose to update the index whenever a new file is added, or it might wait to collect a number of files before indexing them all together at a time when the server load is low. Specifying responsibilities in terms of outcomes also means that an agent can change its own behavior (e.g., improving its performance through learning) and still satisfy the responsibility. As with conditions, when new agents or agent roles are added to the system, there may be better ways available to implement the agent role. So, we do not want to constrain how this responsibility is implemented. We leave the details of how it is done to the agent class specification.

Additionally, because an "Outcome" is usually visible, specifying responsibilities in terms of outcomes instead of behaviors enables auditing of agent behavior. As agents may perform tasks in a large variety of ways, it would be very difficult to determine whether an agent is fulfilling an agent role properly by observing the steps that the agent takes to achieve the task. "Outcomes" are easier to observe and verify. The ability to audit agent behavior is an important property of open agent systems; new, unfamiliar agents may be added to the system.

The "Constraint" allow us to specify requirements on how the responsibility is performed. In the Indexer example above, we specified a constraint on how long before the index needed to be updated. Agents may perform better than required. The constraints are the minimal performance accepted while still satisfying the agent role. They enable us to specify performance requirements on responsibilities.

**Initiatives**

Initiatives specify pro-active behaviors. These are specified as outcomes that the agent playing this agent role must ensure are true. We specify these as Outcomes with Constraints. Specifying in terms of Outcomes has the same benefits as it did for Responsibilities. The format of an Initiative is as follows:

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>No invalid entries are in the index</td>
<td>For longer than a week</td>
</tr>
</tbody>
</table>

Again, using the Indexer agent role described earlier, a possible Initiative for that agent role may be:

This Initiative specifies that the agent playing this agent role must ensure that invalid entries do not remain in the index for longer than a week.

We specify this as an Initiative as there are perhaps no specific events that the agent can detect to ensure that the index hasn't been corrupted. (For example, users may have used a "back door" method to delete or modify files without the indexer agent knowing it.) We keep this flexible in the agent role specification, and leave it to the agent classes to specify how to ensure this holds. For example, the agent may be able to do nightly audits, or it may regularly sample entries in the index and verify that they are still valid.
Facilities

Facilities are requests that other agents or users can ask of the agent playing this agent role. The agent can decide when/if/how they handle the request. A request may fail if the agent is unable to perform the task. Facilities can be for many types of requests, for example, requests for a second opinion on something. (Perhaps an agent can seek the opinion of a number of agents to help it make a decision.)

We specify Facilities as follows:

Requested-Outcome (Parameters) : Specifiable-Constraints

Using the Indexer agent role as an example, a Facility for that agent role may be:

Subdirectory index updated (Subdir-Name) : Completion-Time

This Facility states that we can ask the agent performing the "Indexer" agent role to update the index for all the files in the directory specified by the parameter "Subdir-Name". It also states that the request can include a "Completion-Time" constraint to specify how long the agent can take to complete the task. For example, we may specify that the request be completed within the hour. (The request may fail if the agent is unable to satisfy this constraint.)

As with Responsibilities and Initiatives, we specify the desired Outcome, but not how the Outcome is achieved. The details of how it is achieved are left to the agent class specification.

4. Agent Class Specifications

Agent classes specify one particular way of fulfilling the requirements of an agent role. They are derived during the architectural design phase. While agent role specifications deliberately avoid implementation bias, we begin to capture design decisions in agent class specifications. Agent class specifications describe how the functionality described by the agent role specification can be implemented. This description includes a high-level description of how each role specification element will be implemented, and also which other agent roles the agent class needs to interact with.

In order for an agent to play an agent role, an agent will implement one of the agent role’s agent classes. This provides the agent with a particular way of playing the agent role. In some advanced agents, it is possible that they implement multiple agent classes for an agent role. This enables the agent to play the agent role in a number of different ways. The agent can choose the best way for the given situation.

Agent class specifications are an extension of agent role specifications, and also make use of Responsibilities, Initiatives and Facilities. However, agent class specifications also include additional information not found in agent role specifications. Agent class specifications contain:

- Agent roles fulfilled by this agent class
- Agent roles used by this agent class
- Knowledge
- Public role specification elements with implementation descriptions
- Private role specification elements with implementation descriptions

In this section, we provide descriptions of these agent class concepts. We will then illustrate them in Section 5 using an example. The method and format for specifying agent classes will be described in the example.

Agent Roles Fulfilled

Agent classes specify a particular way of fulfilling an agent role. This specifies which agent roles this agent class fulfills.

Agent Roles Used

An agent class may use the functionality offered by other agent roles in order to implement its role specification elements. In effect, the agent class specifies agent role positions that it needs other agents to fill. (The concept of role positions here is similar to that described in [8].) These positions can be filled by any agent that plays the required agent role. This provides support for agent polymorphism. Note that an agent class only interacts with other agents via their agent roles.

Because the agent roles used can be described using agent classes, which in turn may use other agent roles, defining the agent roles used in an agent class is similar to a hierarchical breakdown where we specify more detail with each iteration. This hierarchical relationship also implicitly defines groups of agents that work together. The group works together to fulfill an agent role.

The details of how it uses other agent roles are contained in the implementation descriptions of its role specification elements. However, to make the relationships visible, agent class specifications also include a list of agent roles used by the agent class.

Knowledge

Knowledge is storage for an agent class. It is analogous to member variables in OO. Knowledge was not needed in agent roles as they specified the features an agent will present to others, and details about the internal storage of the agent was irrelevant. However, for agent classes, we describe how the role specification elements will be
implemented. To describe this, we need to specify what knowledge the agent will store. The details of how the knowledge is stored is a point for detailed design, so at the architectural design level, we describe what data is stored without specifying how it is stored. In the example below, we describe the agent class knowledge using plain text.

**Public and Private Role Specification Elements**

In agent classes, we specify role specification elements as public or private. Public role specification elements are the ones that are accessible by other agent classes. The role specification elements from the agent roles fulfilled by the agent class should appear as public role specification elements. Private role specification elements are those that are defined purely for internal use by the agent class; they cannot be accessed by other agents. These are like the “hidden” implementation details of the agent class, and are analogous to private methods in OO classes.

**Implementation Descriptions for Role Specification Elements**

For agent class specifications, we also include implementation descriptions for each role specification element. (We specify these for both public and private role specification elements.) These describe how the agent class interacts with other agent roles, and explains how the agent class makes use of these other agent roles. These descriptions are like high level design descriptions that discuss how the agent class interacts with other agent roles. The implementation descriptions can be written as text, as UML interaction diagrams, or as Agent UML protocol diagrams [9]. The example in Section 5 uses plain text to describe the interactions.

5. An Example of Agent Role and Agent Classes

To illustrate the concepts of agent roles and agent classes, we will work through an example of how we would specify a software system using agent roles, and how we would step-wise refine it into an architectural design consisting of agent classes. This example will help demonstrate how agent roles and agent classes are specified.

The example system we will use is QuizHelper, an agent-based application that operates in conjunction with CoursesOnline (http://www.coursesonline.com.au), an online education website developed by Hearne Scientific Software (http://www.hearne.com.au). QuizHelper was developed about a year ago, and was created without the use of agent roles and agent classes. However, as its requirements are well defined, we take this opportunity to revisit it and see what kind of design would emerge by considering it in terms of agent roles and agent classes.

**Overview of QuizHelper**

QuizHelper enhances CoursesOnline’s online quiz functionality by providing directed feedback to students after they complete quizzes. For any question that a student answers incorrectly, QuizHelper provides the student with hypertext links to course materials that the student can use to improve their understanding. The aim is to provide immediate and customised feedback that will encourage and help students to learn from their mistakes.

QuizHelper automatically analyses the quiz questions and determines which course materials are relevant for each question. The course author does not need to specify the relevant resources manually, nor provide additional information about the quiz questions and course materials. This enables QuizHelper to work with existing courses that do not know about QuizHelper, and means that course authors do not need to do any extra work to use QuizHelper with their courses.

![Fig 2. Quiz results from CoursesOnline](image)

**Defining the QuizHelper Agent Role**

We can represent the requirements of QuizHelper as a single agent role. (For other systems, the requirements may need to be specified using a set of agent roles.) From a user’s perspective, the QuizHelper agent role needs to provide relevant course material links for each quiz question. We do not consider how the QuizHelper agent will get these links as that is a design issue. We specify the agent role as follows:
**Quiz Resource Manager Role**

**Facilities:**
- Provide a prioritized list of relevant lesson materials for the specified quiz question.

This agent role has a single facility. This facility enables users to request a prioritized list of resources for an incorrectly answered question and display it to the user. The CoursesOnline system can use this facility to retrieve the information it requires for the QuizHelper functionality.

**Specifying the QuizHelper Agent Class**

Next, we specify an agent class for the QuizHelper agent role. We will create a “Quiz Resource Manager Class” that fulfills the “Quiz Resource Manager Role”. The Quiz Resource Manager Class describes one specific way in which the Quiz Resource Manager Role can be implemented. In determining this specific way, we need to make some design decisions.

The “Quiz Resource Manager Class” needs to fulfill the facility specified in the “Quiz Resource Manager Role”. It needs to be able to return relevant lessons materials for a given quiz question. We note the following:

- The list of relevant resources for each question cannot be generated from scratch very quickly. Generating this list involves analyzing the quiz questions, and searching the lesson materials for relevant resources. The overhead of accessing and examining the course materials makes this difficult.
- To speed up the processing, we want an index of the course materials that we can search quickly. This index can be built and maintained at any time.
- To speed up the processing, we can also analyze the quiz questions in advance, and store this information. (This also saves re-analyzing questions when the course materials have been updated, but the questions have not been updated.)

To handle these issues, our “Quiz Resource Manager Class” will use two supporting agent roles. (Note: Using these agent roles is a design decision. It would be possible to specify agent classes that fulfill the “Quiz Resource Manager Role” without using these supporting agent roles.)

We define the following supporting agent roles:

**Lesson Material Manager Role**

**Facilities:**
- Provide a prioritized list of locations in the course materials where a word occurs.
- Provide a prioritized list of locations in the course materials where a phrase occurs.

**Quiz Material Manager Role**

**Facilities:**
- Provide a prioritized list of keywords from a quiz question.

**Responsibilities:**
- When new Quiz Question Information is Available -> All Agents are Notified

The Lesson Material Manager Role is responsible for providing information about the course materials. It will also notify other agents when new course materials are available.

The Quiz Material Manager Role is responsible for providing information about the quiz questions. It will also notify other agents when new course materials are available.

The intention is that the Quiz Resource Manager Class will implement its facility by requesting question keywords from the Quiz Material Manager Role, and then ask the Lesson Material Manager Role to search on those keywords. The Quiz Resource Manager Class can then process and return the results.

So, the Quiz Resource Manager Class will make use of the two agent roles that we have specified. We can consider these supporting agent roles as details of the Quiz Resource Manager Class. As we specify more and more supporting agent roles, we are adding design details in a manner that is similar to structured design.

Further more, to speed up the retrieval, we will design the Quiz Resource Manager Class to pre-process and store an index of relevant lesson materials for each quiz question. This way, the Quiz Resource Manager Role’s facility can be implemented by simply retrieving results from the index and returning them. This design means that this internal index needs to be kept up to date.

Now that we have considered how the Quiz Resource Manager class will operate, let’s look at how we specify the Quiz Resource Manager agent class.
Quiz Resource Manager Class
fulfills Quiz Resource Manager Role
uses Quiz Material Manager Role, Lesson Material Manager Role

Public:
Facilities:
• Provide a prioritized list of relevant lesson materials for the specified quiz question.
This information is retrieved from the index of relevant lesson materials for each quiz question, and returned.

Private:
Knowledge:
• Index of relevant lesson materials for each quiz question.

Responsibilities:
• Course materials changed -> Index of relevant lesson materials for each quiz question is updated.
  1. Lesson Material Manager Role (LMMR) notifies this agent class that the lesson material index is updated.
  2. For each quiz question, Quiz Resource Manager Class (QRMC) uses the Quiz Material Manager Role’s (QMMR) facility to retrieve the list of significant keywords for each question.
  3. For each quiz question, QRMC uses the LMMR’s facility to request the relevant lesson materials for the given question keywords.
  4. QRMC uses the retrieved information to update its Index of relevant lesson materials for each quiz question.

• Quiz questions changed -> Index of relevant lesson materials for each quiz question is updated.
  1. Quiz Material Manager Role (QMMR) notifies this agent class that quiz questions have been updated.
  2. QRMC uses the QMMR facility to retrieve the list of significant keywords for the modified questions.
  3. QRMC uses the LMMR facility to request the relevant lesson materials for the modified questions.
  4. QRMC uses the retrieved information to update its index of relevant lesson materials for the modified quiz questions.

The agent class specification first lists the name of the agent class. It then follows with the list of agent roles fulfilled by this agent class, and the list of agent roles used by this agent class.

It then specifies its public role specification elements. The public role specification elements include the role specification elements of the agent roles that the agent class fulfills. In this case, we include the single facility specified in the Quiz Resource Manager Role. We also describe how this facility will be implemented. In this case, we retrieve the information from the agent class’s internal index and returns it.

Next, we describe the knowledge stored by the agent class. The agent class maintains an index of relevant lesson materials for each quiz question. This information is returned to the user by the agent class’s public facility (described earlier). This index needs to be created and maintained by the agent class. The functionality to do this is described in the agent class’s private role specification elements.

Finally, we describe the agent class’s private role specification elements. For this agent class, we specify two responsibilities that create and maintain the agent class’s internal index. One responsibility updates the index when the course materials change. The other updates the index when the quiz questions change. In the implementation description for these responsibilities, we describe the dynamic behavior of these responsibilities. We describe the agent roles that the agent classes interact with, and describe the sequence in which these interactions occur. As mentioned earlier, it is possible to specify these implementation descriptions using UML interaction diagrams or Agent UML protocol diagrams.

This completes the agent class specification. Agent class specifications are at the architectural design level. The agent class specification describes how the agent class fulfills the role. It describes how the agent class interacts with other agents, and what roles those other agents need to play. The role specification elements’ implementation descriptions outline the sequence in which interactions occur, and with whom they occur.

Specify Agent Classes for the “Supporting” Agent Roles

When specifying the “Quiz Resource Manager Class” agent class, we specified two supporting agent roles. In this step, we break the design down into more details by specifying agent classes for these supporting agent roles.

As with the Quiz Resource Manager agent class, we make “public” the role specification elements of the agent roles being fulfilled. We then add knowledge, and private role specification elements. The implementation descriptions of the role specification elements describe how those elements are implemented.

The Lesson Material Manager Class and Quiz Material Manager Class operate in a similar fashion to the Quiz Resource Manager Class. They maintain an internal index of data about the lesson materials and quiz questions respectively, and their facilities return information from
that index to ensure that they can be performed in a timely manner.

Due to their similarities to the Lesson Material Manager Class, we will not explain them in detail. We present the agent class specifications for the Lesson Material Manager Class and Quiz Question Manager Class:

**Lesson Material Manager Class**

*fulfills Lesson Material Manager Role*

**Public:**

- **Facilities:**
  - Provide a prioritized list of locations in the course materials where a word occurs.
  - Search the Index of Lesson Materials that is maintained by this agent class, prioritise the results, then return them.
  - Provide a prioritized list of locations in the course materials where a phrase occurs.
  - Search the Index of Lesson Materials that is maintained by this agent class, prioritise the results, then return them.

- **Responsibilities:**
  - When new Course Material Information is Available ->
    All Agents are Notified
    Broadcast a notification to all agents each time the Index of Lesson Materials is updated.

**Private:**

- **Knowledge:**
  - Index of lesson materials. (Indexed to support word and phrase searching.)

- **Responsibilities:**
  - Course Materials Changed ->
    Index of lesson materials is updated to match.
    1. CoursesOnline broadcasts an event when lesson materials are added, removed or modified.
    2. The Lesson Material Manger Class receives the notification, and requests the changed lesson materials from CoursesOnline.
    3. The Lesson Material Manager Class then indexes the changed materials, and updates its internal Index of lesson materials appropriately.

**Quiz Material Manager Class**

*fulfills Quiz Material Manager Role*

**Public:**

- **Facilities:**
  - Provide a prioritized list of keywords from a quiz question.
  - Search the Index of Quiz Keywords that is maintained by this agent class, and return the keywords for the quiz question.

- **Responsibilities:**
  - When new Quiz Question Information is Available ->
    All Agents are Notified
    Broadcast a notification to all agents each time the Index of Quiz keywords is updated.

**Private:**

- **Knowledge:**
  - Index of Quiz keywords (list of prioritized keywords for each quiz question).

- **Responsibilities:**
  - Quiz Questions Changed ->
    Index of Quiz keywords is updated to match.
    1. CoursesOnline broadcasts an event when quiz questions are added, removed or modified.
    2. The Quiz Material Manger Class receives the notification, and requests the changed quiz questions from CoursesOnline.
    3. The Quiz Material Manager Class then analyses the question and answer text, and updates its internal Index of Quiz keywords appropriately.

At this stage, we have specified a number of agent classes, and specified how they interact. Together, this set of three agent classes implements the QuizHelper functionality. This is an agent-oriented architectural design.

As mentioned earlier, the QuizHelper system was initially developed without the use of agent roles and agent classes. Comparing the design that we have developed here to the one that was implemented, the one described here is cleaner and more cohesive. In the original design, there was a lot of coupling between the agents, and it was not always clear why certain design decisions should be made. Designing with agent roles and agent classes enables a clean separation of requirements and design. Agent roles capture requirements information, while agent classes capture design information. This separation helps to ensure that the interfaces are cohesive, encouraging less coupling in the design.
6. Discussion

The concepts of agent roles and agent classes specified in this paper are a natural extension of OO concepts for agent-based systems. (They map to OO concepts more readily than the concept of roles as defined in other AOSE methodologies.) As mentioned earlier, agent roles are analogous to OO interfaces, while agent classes are analogous to OO classes. This similarity may make it easier for current practitioners of OO to migrate to AO development. Easing this transition is critical if AO methodologies are to become more widely used.

While there is an analogy to OO, agent roles and agent classes add new features not currently found in OO. In effect, we are replacing the member variables and operations of OO with responsibilities, initiatives, facilities, and knowledge. These elements allow us to specify agents as active objects.

Objects are passive. They will not generally do anything unless one of their operations is called. Agents on the other hand are active. Responsibilities and Initiatives enable an agent to take action even when they have not been explicitly asked to do so. They can monitor and react to changes in the environment, or proactively take action to prevent undesirable outcomes. Facilities are similar to OO operations. However, facilities have more autonomy than operations in that they can choose when and how they do their job. (An facility can be acted on after a delay provided that there isn’t a time constraint specified that requires it to be performed immediately.)

As agent roles’ responsibilities, initiatives and facilities are specified in terms of outcomes, we leave it to the agent to determine how these outcomes are achieved. This flexibility enables agents to improve their performance (e.g. through learning the user’s preferences) and change the way they operate. Even though an agent’s behavior may change, agent roles provide bounds to ensure that the agents are functioning as expected.

Designing agent-based systems with agent roles and agent classes also lends itself to supporting verification and validation activities, an important aspect of any software engineering methodology. As we saw in section 5, agent role specifications provide a requirements level view of the system. It is not difficult to verify that the requirements have been translated accurately to agent role specifications. In fact, it may be possible that our familiarity with complex, role-based, human systems (e.g. a company) may enable us to understand a system of roles more easily than a system of classes. Similarly, the mapping from agent roles to agent classes can also be verified easily. We can verify that each role specification element is implemented correctly, and that they work together correctly. The ability to verify that an agent-based architectural design matches our initial requirements is an important attribute of an AOSE methodology.

The use of agent roles and agent classes enables a clearer separation of requirements and design than other AOSE methodologies. (e.g. Gaia [13], ROADMAP [6]). In Gaia and ROADMAP, roles are specified using liveness and safety expressions. These expressions describe the behaviors of roles in a procedural manner. They describe the order in which actions occur, and where in the sequence it interacts with other roles. This is a design-level description as it makes decisions on how the roles will be implemented. (This is the kind of information contained in our agent class specifications.) These methodologies lack a clear requirements-level concept such as the agent roles described in this paper.

7. Conclusions and Further Work

In this paper, we have discussed the concepts of agent roles and agent classes. These concepts can be applied to a wide variety of application domains.

This work was supported by the Smart Internet CRC. We plan to run further case studies in applying these concepts to the specification and design of agent-based systems. We will refine these concepts based on these studies. When the concepts have been tested, we will build an AOSE methodology around these concepts.

8. References


