Specifying Roles within Agent-Oriented Software Engineering

Kevin Chan Leon Sterling
University of Melbourne
{kchan,leon}@cs.mu.oz.au

Abstract

Roles are an essential concept within agent-oriented software engineering (AOSE). Role definitions in current AOSE methodologies are usually claimed to be for use at the requirements level. However, in most methodologies, they are too low level, specifying too much detail. In this paper, we present a "higher level" role specification. The role specification method described in this paper works together with other agent specification/analysis methods that we and others have developed. However, we believe that role specification may also be used with non-agent-based systems, and provide a useful abstraction for specifying the requirements of any software system.

1. Introduction

Software agents have been an active area of research for many years. In the last few years, agent-oriented software engineering (AOSE) has become an active sub-area. AOSE is focused on methodologies for creating predictable, reliable, and stable agent-based systems. A number of methodologies have been proposed in recent times, including Gaia [13], MESSAGE [15], ROADMAP [6], Tropos [3] and Prometheus [10]. There has also been a wide variety of work on techniques and concepts to support AOSE methodologies (e.g., [9], [12], [1], [5], and [8]).

In this paper, we focus on role specifications. Roles are an important part of many current AOSE methodologies. Role specifications are the first artifacts created by many methodologies. Most current AOSE methodologies [13] [6] [10] agree that role specifications are useful during the requirement analysis or gathering phases of developing agent-based systems. However, the role specifications in most AOSE methodologies contain too much implementation bias for the requirements phase. In this paper, we describe a new role specification method that removes the bias, enabling the method to be applied more easily during the requirements phase.

Our described role specification method may also be useful for analyzing/capturing the requirements of non-agent-based systems. In some ways, our role specifications are at a similar level of abstraction to UML Use Cases. People have long been designing complex human systems (for example, a company) in terms of interacting, intelligent agents. Using role specifications to describe software systems may enable modelers to leverage off this familiarity, to conceptualize and describe the software system with greater ease.

2. Conceptualising Roles

Roles are present in human organizations. For example, roles at a university include "Head of Department" and "Lecturer". Roles may be performed by more than one person (e.g. there are many lecturers at a university), and one person may simultaneously take on multiple roles (e.g. one person may be both a head of department and a lecturer). Roles define a person's responsibilities and function in an organization. Roles also define how people interact with each other (e.g. Students would ask a lecturer for lecture notes. They would not ask the head of department.) Roles in agent-based systems are much the same. They define the responsibilities and function of agents in an agent-based system.

In this paper, the role specification method we propose caters for agent-based systems in which:

- Agents are objects at the granularity of a process.
  - Agents have their own thread of processing
- Agents in the system can be heterogeneous
  - Agents can be implemented in different languages and using different architectures
- Agents exist in an open, dynamic system
  - Agents can enter/exit the system
  - New, unfamiliar agents can be introduced
  - As new agents may be introduced into the system, we need to be able to query agents to determine their abilities
  - To ensure new/unfamiliar agents behave as expected, agent behavior should be auditable
- Interactions between agents can change at run time
  - For example, agents may start interacting with new agents who have recently joined the system
- Agents can change their behavior during run time

  - For example, agents may use learning to modify their behaviors to improve their performance.

Additionally, role specifications should be informal, but structured. As [14] points out, "if agent-oriented techniques are ever to become widely used outside the academic community, then informal, structured methods for agent-based development will be essential". The role specifications we describe are of the formality of UML, the de facto standard for modeling object-oriented systems). (There has been work on Agent UML, which adapts UML for use in specifying agents [9] [1] [5]. However, for the types of agents we are considering, these specifications are more suited for specifying architectural design and not for requirements.)

We propose specifying roles using three levels of description (See Figure 1):

- Role Specification: This model is at the requirements level. It specifies what the role will do without specifying how the role will do it. This is like the "interface" for an agent. An agent can fulfill multiple role specifications simultaneously.
- Role Design Specification: This model is at the early architectural design level. This specifies a way for the agent to implement the role. This includes describing which other roles it needs to interact with in order to implement the role. Note that there can be more than one role design specification for a given role specification. i.e., there may be more than one way to fulfill the role. An agent may implement multiple role design specifications simultaneously for a single role specification. This would enable the agent to choose and apply the role design that is most suitable for the current situation. Or, a system may have a number of agents that fulfill the same role, but with each agent using a different role design. In this case, the agent that is currently most suited to fulfill the role does the job.
- Role Implementation Specification: This model is closely related to the Role Design Specification, and is also at the architectural design level. This holds information about the agent design specification that is specific to an agent framework/environment. For example, the details about the format of agent messages, and details about parameter types. Note that this model does not contain details about the agent design (e.g. agent's internal cognitive model, algorithms, etc.). This model is intended as a data dictionary for storing framework-specific details about role design specifications, allowing these details to be removed from the role design specifications. The information is useful for determining how to build new agents that can work with existing agents.

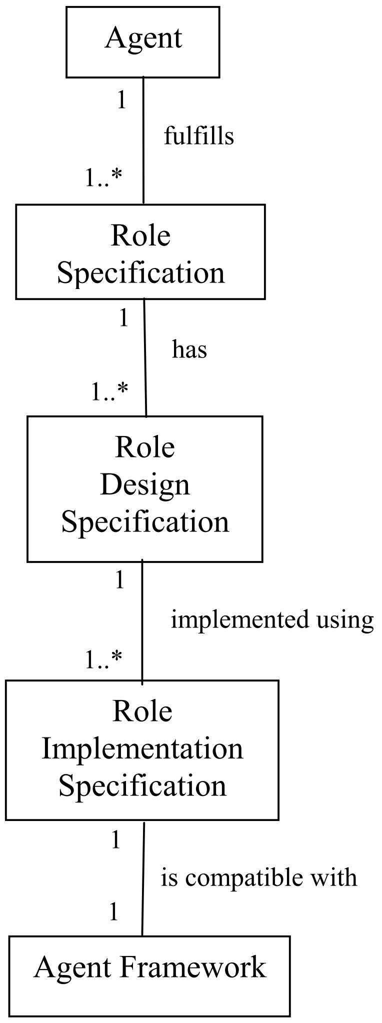


Figure 1. Three Levels of Role Specification.

We start by creating Role Specifications, then use them to derive Role Design Specifications and Role Implementation Specifications. The three models work together to take us from the requirements phase to the early stages of architectural design. We specify the functionality of the roles, ways in which the roles can interact to achieve their purpose, and some details about the communication between roles. These models can be used as an input to the architectural design phase of development. The system could eventually be implemented using an agent-oriented language such as JACK [4], or a more common object-oriented language such as Java.

While the three models are designed to work together, this paper focuses on the Role Specification model and how it is used during the requirements phase.

3. Role Specification

Role specifications specify the functionality of roles and are derived during the requirements phase. Role specifications are focused on what the role does, but does not describe how these tasks should be achieved. We do not want to describe how roles are fulfilled as that is a design decision. They are in some ways like "interfaces" for agents. Agents interact with each other via their roles.

Complex human organizations (such as a university or a company) are made up of interacting, intelligent agents. As we are familiar with these organizations and how they operate, specifying complex software systems as societies of interacting roles may be a natural metaphor that enables us to specify software systems more easily.

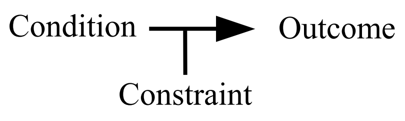
We specify Role Specifications in terms of:

- Responsibilities: Specify what the role will do in reaction to changes in the environment. Most of a role's functionality can be specified as responsibilities.
- Initiatives: Specify 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 this role. These are like object-oriented methods, except that agents have autonomy meaning that the agent can decide whether it will perform the request.

Role specifications should be as cohesive as possible. The Responsibilities, Initiatives and Facilities specified for a 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 fulfill a set of roles. We should not specify bloated roles for these cases.

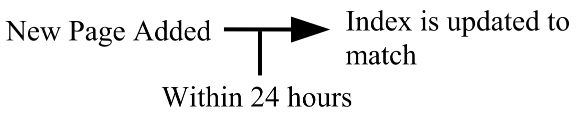
3.1 Responsibilities

Responsibilities are specified as follows:



This describes the condition for activating the responsibility, the outcome that the agent's behavior should achieve, and any constraints on this responsibility.

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



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 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/roles are introduced into the system. The role specification should be flexible enough to support future enhanced implementations that cannot be implemented currently. We leave the details of how it's done to the role design.

We specify the expected "Outcome" required in response to the "Condition" rather than the behavior required. Again, we want to keep this flexible in the role specification. An agent may achieve the desired outcome in a number of different ways, we do not want to constrain how an agent will achieve the required outcome. Using the example from before, an agent implementing the Indexer 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. Also, specifying Responsibilities in terms of outcomes also means that an agent can change its own behavior (e.g., through improving its performance through learning) and still satisfy the Responsibility. As with conditions, when new agents/roles are added to the system, there may be better ways available to implement the role. Again, we leave the details of how it's done to the role design.

Additionally, an "Outcome" is usually visible, and 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 its 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 role.

3.2 Initiatives

Initiatives specify pro-active behaviors. These are specified as outcomes that the agent 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:

Outcome
|
Constraint

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

No invalid entries are in the index
|
For longer than a week

This Initiative specifies that the role should ensure that invalid entries should 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 role specification, and leave it to the role design to specify how to ensure this holds. For example, the agent may be to do nightly audits, or it may regularly sample entries in the index and verify that they are still valid.

3.3 Facilities

Facilities are requests that other agents or users can ask of this 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 role as an example, a Facility for that role may be:

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

This Facility states that we can ask the agent performing the "Indexer" 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 role designs.

4. Discussions and Related Work

In this section, we discuss the role specifications used in three current AOSE methodologies: Gaia, ROADMAP and Prometheus. We examine these methodologies in particular because roles play an important part in them.

4.1 Gaia Roles

In Gaia [13], roles are specified and refined during the requirements analysis phase. They are intended to describe the requirements of the system. Gaia roles are defined by four attributes: responsibilities, permissions, activities, and protocols. We focus on the responsibilities.

The responsibilities attribute in the primary way through which Gaia specifies the functionality of roles. Responsibilities are specified in terms of:

- Liveness properties: "states of affairs that an agent must bring about, given certain environmental conditions" [14]
- Safety properties: "invariants" that the agent must maintain. The agent must ensure "that an acceptable state of affairs is maintained across all states of execution" [14]

In some ways, liveness properties are similar to the "Responsibilities" in our method, while safety properties are similar to the "Initiatives" in our method.

Safety properties are expressed as predicates. For example:

coffeeStock > 0

This means that the agent must ensure that the coffee stock is always greater than 0. Safety properties are specified at a similar level of detail to "Initiatives" in our method.

Liveness expressions in Gaia are specified using "liveness expressions", which are based on the life-cycle expressions of FUSION. An example liveness expression [13]:

CoffeeFiller
= (Fill**.**InformWorkers**.**<u>CheckStock</u>**.**AwaitEmpty)$^{\omega}$

This expression says the CoffeeFiller role consists of executing the protocol Fill, followed by the protocol InformWorkers, followed by the activity CheckStock, followed by the protocol AwaitEmpty. The "ω" denotes that this sequence of protocols and activities is repeated to infinity.

Protocols are defined in Gaia's Protocol Model. Protocols are described with the attributes:

- Purpose: What the protocol does
- Initiator: Role(s) that starts the interaction
- Responder: Roles(s) that the initiators interact with
- Inputs: Information used by the role initiator
- Outputs: Information supplied by/to the responder during the interaction
- Processing: Brief textual description of any processing that protocol initiator performs during the interaction

Protocols "define the way that it [a role] can interact with other roles" [14]. While protocols are more complex than object-oriented method calls (with dialogues between agents, etc), protocols are defined almost at the level of method calls. (They describe inputs, outputs and processing.) So, roles in Gaia are described at a procedural level. They prescribe the steps that an agent takes when fulfilling a role. These role specifications do not support the flexibility of allowing multiple ways of fulfilling a role. The information in the Gaia role model is similar to the information contained in our "Role Design" model.

4.2 ROADMAP Roles

ROADMAP [6] builds on Gaia. It adopts the liveness and safety expressions used in Gaia. As a result, its role definitions also contain more design information than there should be.

In the ROADMAP meta-model [7], the concept of evaluation functions is added to ROADMAP role specifications. Evaluation functions represent performance qualities that we want the role to exhibit. For example, "Reliability() > 8" specifies that the agent must ensure the value of the evaluation function "Reliability()" is kept above 8. This concept is similar to the constraints specified in our role specifications.

4.3 Prometheus Roles

In Prometheus [10], "roles" are referred to as "functionalities". Prometheus functionalities are described as follows:

- Name
- Short natural language description
- List of actions
- List of relevant percepts
- Data used
- Data produced
- Interactions with other functionalities

The description of "interactions with other functionalities" specifies what message the role sends and receives from other roles. This constrains how the role interacts with other agents, and constrains which other roles the given role can interact with. In effect, this describes one particular way in which the role can be fulfilled. In our role specification method, this information is described in the Role Design Specification.

However, Prometheus functionalities do not specify the details of how an agent should fulfill a role. So, Prometheus functionalities have less implementation bias than role specifications in Gaia and ROADMAP.

The role specifications defined in the methodologies discussed are more similar to our Role Design Specifications than our Role Specifications. A reason for this is that the role specifications in these methodologies were not created with the requirements for dynamic, open systems that we specified in section 2. The Role Specification model we described has less implementation bias than the role specifications used in these methodologies. Reducing implementation bias makes the method more effective as a requirements tool as we should not be specifying design information during the requirements phase.

5. Conclusions and Further Work

In this paper, we described a new way of specifying agent roles, and examined how roles are specified in other, current AOSE methodologies. The method described contains less implementation bias than other AOSE methodologies, making the method more suitable for use during the requirements phase.

We also described how the Role Specifications described are used in conjunction with Role Design Specifications and Role Implementation Specifications mode in order to lead the modeler from the requirements phase to early architectural design. We will describe the other two models in more detail in a future paper.

The role specification method described in this paper is currently being used for a project with the Smart

Internet Cooperative Research Centre. We intend to refine the method as we apply it to more projects.

6. References

[1] Bernhard Bauer, Jörg P. Müller, James Odell, "Agent UML: A Formalism for Specifying Multiagent Interaction", *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103, 2001.
[2] Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Languge User Guide,* Addison-Wesley, 1999.
[3] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia and John Mylopoulos, "A knowledge level software engineering methodology for agent oriented programming", Proceedings of the Fifth International Conference on Autonomous Agents, ACM Press, Montreal, Canada, Jörg P. Müller and Elisabeth Andre and Sandip Sen and Claude Frasson", pp. 648--655, 2001.
[4] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas, "JACK Intelligent Agents – Components for Intelligent Agents in Java", Technical Report, Agent Oriented Software Pty. Ltd., Melbourne, Australia, 1998.
[5] M. Huget, "Agent uml class diagrams revisited", *Technical Report ULCS-02-013*, Department of Computer Science, University of Liverpool, 2002. http://citeseer.nj.nec.com/article/huget02agent.html
[6] Juan, T., Pearce, A. and Sterling, L., "ROADMAP: Extending the Gaia Methodology for Complex Open Systems", *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, p3-10, Bologna, Italy, July 2002.
[7] Juan, T. and Sterling, L., "A Meta-Model for Intelligent Adaptive Multi-Agent Systems in Open Environments" (poster), Proceedings 2nd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Melbourne Australia, July, 2003.
[8] James Odell, H. Van Dyke Parunak, and Mitch Fleischer, "The Role of Roles in Designing Effective Agent Organizations", *Software Engineering for Large-Scale Multi-Agent Systems*, Alessandro Garcia et al, LNCS, Springer, 2003.
[9] James Odell, H. Van Dyke Parunak, Bernhard Bauer, "Extending UML for Agents", *Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, TX, pp. 3-17accepted paper, AOIS Worshop at AAAI 2000.
[10] Padgham, L. and Winikoff, M., "Prometheus: A Methodology for Developing Intelligent Agents", *Proceedings of the Third International Workshop on Agent Oriented Software Engineering*, at AAMAS 2002. July, 2002, Bologna, Italy.
[11] Lin Padgham and Michael Winikoff, "Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents", *Proceedings of the workshop on Agent-oriented methodologies at OOPSLA 2002*, November 4, 2002, Seattle.
[12] H. Van Dyke Parunak and James Odell, "Representing Social Structures in UML", *Agent-Oriented Software Engineering Workshop II*, Michael Wooldridge, Paolo Ciancarini, and Gerhard Weiss, eds., Springer, Berlin, 2002, pp. 1-16.
[13] Wooldridge, M., Jennings, N. and Kinny, D., "The Gaia Methodology for Agent-Oriented Analysis and Design", *Journal of Autonomous Agents and Multi-Agent Systems,* 3 (3), 2000, 285-312.
[14] M. Wooldridge and P.Ciancarini, "Agent-Oriented Software Engineering: The State of the Art", In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, Springer-Verlag Lecture Notes in AI Volume 1957, January 2001.
[15] Giovanni Caire, Francisco Leal, Paulo Chainho, Richard Evans, Francisco Garijo, Jorge Gomez, Juan Pavon, Paul Kearney, Jamie Stark, and Phillipe Massonet, "Agent oriented analysis using MESSAGE/UML", in Michael Wooldridge, Paolo Ciancarini, and Gerhard Weiss, editors, *Second International Workshop in Agent-Oriented Software Engineering (AOSE-2001)*, pp. 101-108, 2001.
[16] Daniela E. Herlea, Catholijn M. Jonker, Jan Treur, and Niek J. E. Wijngaards, "Specification of Behavioural Requirements within Compositional Multi-agent System Design", *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering ({MAAMAW}-99)*, Volume 1647, Springer-Verlag: Heidelberg, Germany, Francisco J. Garijo and Magnus Boman, pp. 8-27, 1999.