In Agent-Oriented Software Engineering – Proceedings of the First International Workshop on Agent-Oriented Software Engineering, 10th June 2000, Limerick, Ireland. P. Ciancarini, M. Wooldridge, (Eds.) Lecture Notes in Computer Science. Vol. 1957, Springer Verlag, Berlin, January 2001.

An Overview of the Multiagent Systems Engineering Methodology

Mark F. Wood Scott A. DeLoach

Department of Electrical and Computer Engineering
Air Force Institute of Technology
2950 P Street, Wright-Patterson AFB, OH, USA 45433-7765
woodm@stratcom.mil scott.deloach@afit.edu

Abstract. To solve complex problems, agents work cooperatively with other agents in heterogeneous environments. We are interested in coordinating the local behavior of individual agents to provide an appropriate system-level behavior. The use of intelligent agents provides an even greater amount of flexibility to the ability and configuration of the system itself. With these new intricacies, software development is becoming increasingly difficult. Therefore, it is critical that our processes for building the inherently complex distributed software that must run in this environment be adequate for the task. This paper introduces a methodology for designing these systems of interacting agents.

1. Introduction

The advent of multiagent systems has brought together many disciplines in an effort to build distributed, intelligent, and robust applications. They have given us a new way to look at distributed systems and provided a path to more robust intelligent applications. However, many of our traditional ways of thinking about and designing software do not fit the multiagent paradigm. Over the past few years, there have been several attempts at creating tools and methodologies for building such systems. Unfortunately, many of the tools focused on specific agent architectures [1, 12] or have not gone to the necessary level of detail to adequately support complex system development [8, 24]. In our research, we have been developing both a complete-lifecycle methodology and a complimentary environment for analyzing, designing, and developing heterogeneous multiagent systems. The methodology we are developing is Multiagent Systems Engineering (MaSE).

Constructing multiagent systems is difficult. They have all the problems of traditional distributed, concurrent systems, plus the additional difficulties that arise from flexibility requirements and sophisticated interactions. Sycara states in [21] that there are two technical hurdles to the extensive use of multiagent systems. First, there is a lack of a proven methodology enabling designers to clearly structure applications as multiagent systems. Second, there are no general case industrial-strength toolkits that are flexible enough to specify the numerous characteristics of agents.

This paper addresses the first technical hurdle by proposing a methodology for the design of multiagent systems. The focus is on the construction of a multiagent system

through an entire software development lifecycle from problem description to implementation. Research into multiagent system methodologies, for the most part, has focused more on high-level descriptions and concepts than on an actual design methodology. Other design paradigms - object-oriented systems in particular - do exist as general-case solutions, but these are neither tuned for, nor particularly useful in creating a system that is intended to take full advantage of agent capabilities. Object-oriented design has achieved some maturity and provides a stable foundation upon which to build. However, object-oriented methodologies are not directly applicable to agent systems - typical agents are significantly more complex in both design and behavior than objects.

1.1 Scope

Because of assumptions made to simplify the research, MaSE has a few limitations. First, we assume that the system being created is closed and that all external interfaces are encapsulated by an agent that participates in the system communication protocols. Second, the methodology does not consider dynamic systems where agents can be created, destroyed, or moved during execution. Third, inter-agent conversations are assumed to be one-to-one, as opposed to multicast. However, substituting a series of point-to-point messages can be used to fulfill the requirement for multicast. Finally, it is assumed that the systems designed with MaSE would not be very large; the target is ten or less software agent classes. This is not a hard constraint, but simply indicates that no verification or validation of larger systems was done and that no thought was given to the potential problems of such systems.

Work is ongoing at the Air Force Institute of Technology (AFIT) to extend this methodology in these and other areas. Both the problems of dynamic systems and multicast conversations appear to be relatively straightforward extensions using predefined *move* activities and special multicast conversations. While not designed for open systems, MaSE can also be used to design agents that operate in an open environment as long as there are appropriately define protocols for the agent to use.

1.2 Related Work

There have been several proposed methodologies for analyzing, designing, and building multiagent systems [8]. The majority of these are based on existing object-oriented or knowledge-based methodologies. In fact, the syntax of many of the models was taken from the Unified Modeling Language even though the methodology itself is dissimilar to most object-oriented approaches.

Actually, MaSE builds upon the work of many agent-based approaches; it takes many ideas and combines them into a complete, end-to-end methodology. For instance, work on goals and roles by Kendall [11] influenced the initial MaSE analysis steps while the mapping of roles to agent classes builds off the concepts presented by Kinny, Georgeff, and Rao [12]. Only the Gaia approach [24] attempts to encompass the entire life cycle, although the authors admit to its shortcomings. The main advantage of MaSE over previous methodologies is its scope and completeness.

2. Multiagent Systems Engineering Methodology

The Multiagent System Engineering (MaSE) methodology, takes an initial system specification, and produces a set of formal design documents in a graphically based style. The primary focus of MaSE is to guide a designer through the software lifecycle from a prose specification to an implemented agent system. MaSE is independent of a particular multiagent system architecture, agent architecture, programming language, or message-passing system. A system designed in MaSE could be implemented in several different ways from the same design. MaSE also offers the ability to track changes throughout the process. Every design object can be traced forward or backward through the different phases of the methodology and their corresponding constructs. MaSE is described in more detail in [4, 22]. An overview of the methodology and models is shown in Figure 1.

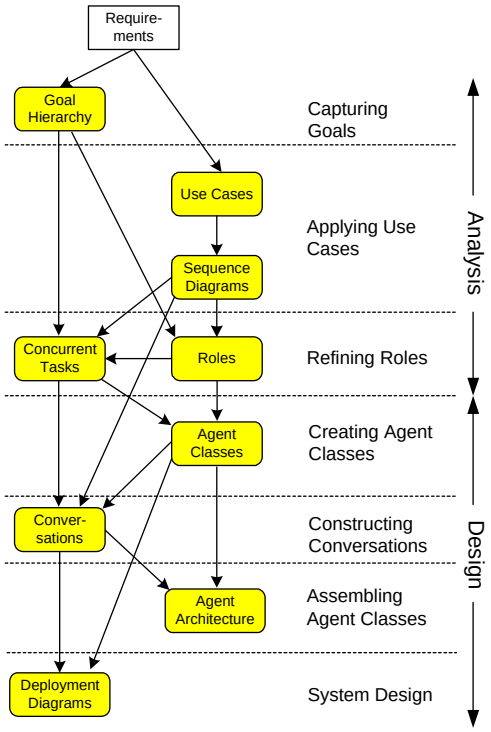

Fig. 1. The MaSE Methodology

The general operation of MaSE follows the progression of steps shown in Figure 1, with outputs from one section becoming inputs for the next. The methodology is iterative across all phases with the intent that successive "passes" will add detail to the models described later. The gray boxes denote models used within the methodology

and the phases are listed down the right side of the figure. The arrows indicate how the models influence each other.

2.1 Capturing Goals

The first phase in MaSE is Capturing Goals, which takes the initial system specification and transforms it into a structured set of system goals as shown in a Goal Hierarchy Diagram (Figure 2). This phase of MaSE is drawn in a large part from analysis patterns in [11]. In the MaSE methodology, a goal is always defined as a system-level objective. Lower-level constructs may inherit or be responsible for goals, but goals always have a system-level context.

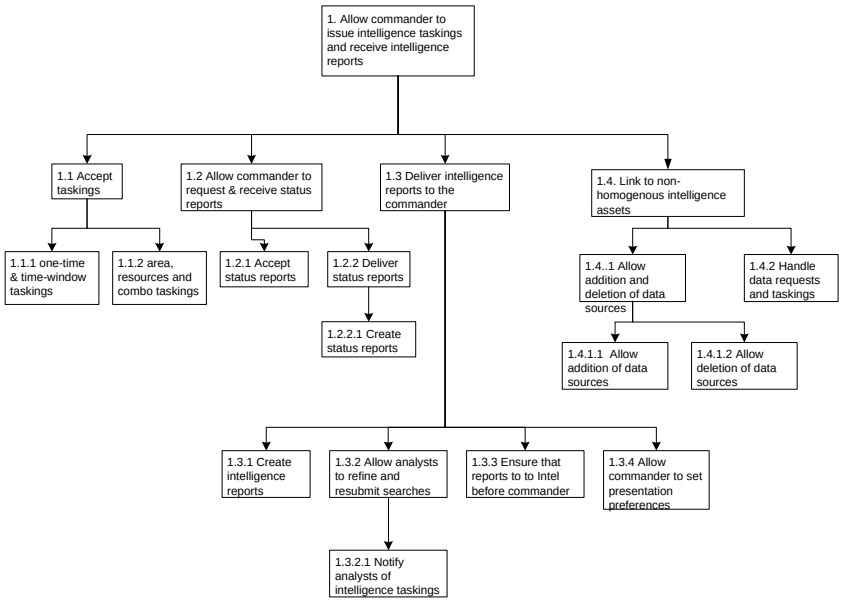

Fig. 2. Goal Hierarchy Diagram

There are two parts of the Capturing Goals phase: identifying and structuring goals. The goals are identified by distilling the essence of the set of requirements. These requirements may include detailed technical documents, user stories, or formalized government specifications. Once these goals have been captured and explicitly stated, they are less likely to change than the detailed steps and activities involved in accomplishing them.

The goals are then analyzed and structured into a form that can be passed on and used in the design phases of the MaSE methodology. In a Goal Hierarchy Diagram, goals are organized by importance. The main sequences of interaction and subordinate details must be distinguishable from one another. Each level of the

hierarchy contains goals that are roughly equal in scope and all sub-goals relate functionally to their parent.

2.2 Applying Use Cases

It is the conversations between agents that are the real backbone of a multiagent system, as they enable the distributed operation that is the strength of agent technology. The second phase of MaSE looks down the road toward constructing these conversations and creates use cases to ease this difficulty.

The Applying Use Cases phase captures use cases from the initial system requirements and restructures them as a Sequence Diagram (Figure 3). A sequence diagram depicts a sequence of messages between multiple agent roles.

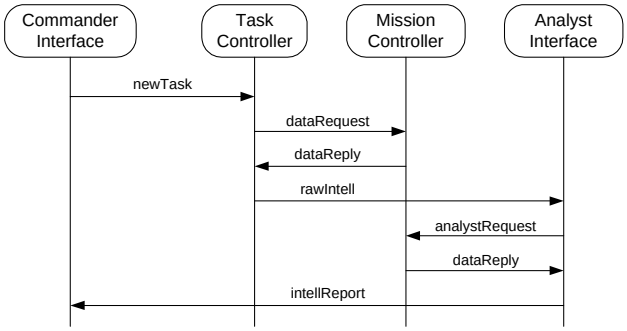

Fig. 3. Sequence Diagram

First, use cases are drawn from the system requirements. Use cases are narrative descriptions of a sequence of events that define desired system behavior. They are examples of how the user (or the requirements document editor) thinks the system should behave in a given case.

A Sequence Diagram is used to determine the minimum set of messages that must be passed between roles. If a message is passed between two roles, then there must be a corresponding communication path between them. A communication path between roles played by separate agent classes means that a conversation must exist between the two agent classes to pass the message. The agent class playing the role that initiated the communication becomes the initiator of that conversation, while the receiving agent class becomes the responder. Typically, we create at least one sequence from a use case. If there are several possible scenarios, multiple Sequence Diagrams are created.

2.3 Refining Roles

The third step of MaSE is to transform the structured goals of the Goal Hierarchy Diagram into a form more useful for constructing multiagent systems: roles. Roles are the building blocks used to define agent's classes and capture system goals during the design phase. We guarantee that system goals are accounted for by ensuring that every goal is associated with a role and that every role is played by an agent class.

A role is an abstract description of an entity's expected function and encapsulates the system goals that it has been assigned the responsibility of fulfilling. Roles are created to do something. They are similar to the notion of an actor in a play or an office within an organization. Roles are described in detail in [10,12,24].

The general case transformation of goals to roles is one-to-one; each goal maps to a role. However, there are many exceptional situations where it is useful to combine goals. Similar or related goals may be combined into single roles for the sake of convenience or efficiency. Goals that share a high degree of cohesion as described in [16] can be combined into a single role.

Some goals imply distributed roles. Any mention of separate machines or other distribution requires one role for each "side" of the distributed relationship. Interfacing with an external source is the same. One role must interface with the source while another may be required to bridge the gap back to the system. This is also true for any database, file interface, or user interface in the system. A user interface implies a role by itself and should be separate from other roles as if it were a separate data source.

Role definitions are captured in a traditional Role Model [10] as shown in Figure 4. MaSE also allows a more complete version of a Role Model, as shown in Figure 5, which includes information on interactions between role tasks. However, the traditional version of the Role Model is more useful at the outset of the role definition process before tasks have been defined, as well as later in the analysis to provide a high-level view of the system. In the traditional Role Model, lines between roles denote possible communications paths between roles. These paths are derived from the Sequence Diagrams developed in the previous step.

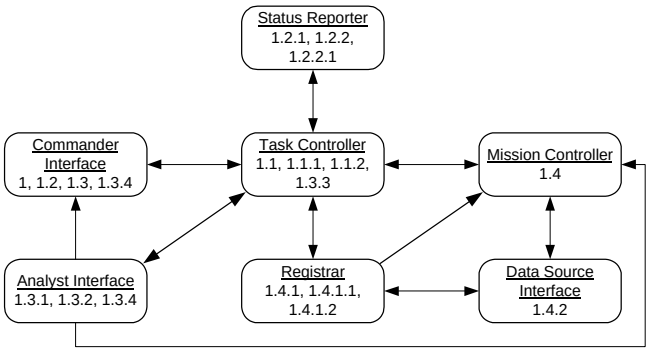

Fig. 4. Traditional Role Model

In MaSE, roles are typically documented in a more detailed version of a Role Model as shown in Figure 5. First, the goals associated with each role are listed under the role name. It also shows the set of tasks associated with each role, which are used to define the role's behavior. Roles are denoted by rectangles, while the role tasks are denoted by ovals attached to the role. Tasks are simply identified in the MaSE Role Model. The detailed description of a task's definition is provided in the next section. Lines between tasks denote communications protocols that occur between the tasks. The arrows denote the initiator/responder relationship of the protocol with the arrow pointing from the initiator to the respondent. Solid lines indicate peer-to-peer communications, which are generally implemented as external communications protocols. External protocols involve message passing between roles that may become actual messages if their roles end up being implemented in separate agents. Dashed lines denote communication between concurrent tasks within the same role. A lined is dashed if it will only occur within the same instance of the role in the final system. Roles may not share or duplicate tasks. Sharing of tasks is a sign of improper role decomposition. Shared tasks should be placed in a separate role, which can be combined into various agent classes in the Design phase.

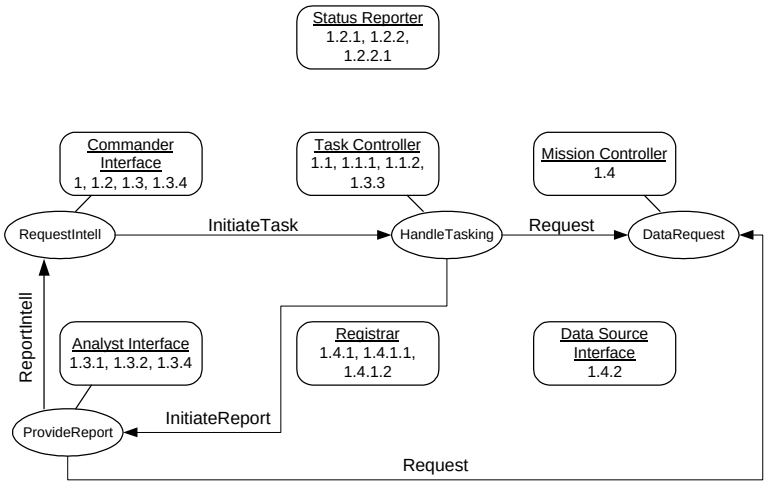

Fig. 5. MaSE Role Model

After roles are created, tasks are associated with each role. Every goal associated with a role can have a task that details how the goal is accomplished. This must be done after role creation since tasks communicate with tasks in other roles. A MaSE task, which captures a bidder's behavior in a Contract Net Protocol, is shown in Figure 6. A task is a structured set of communications and activities, depicted as a state diagram.

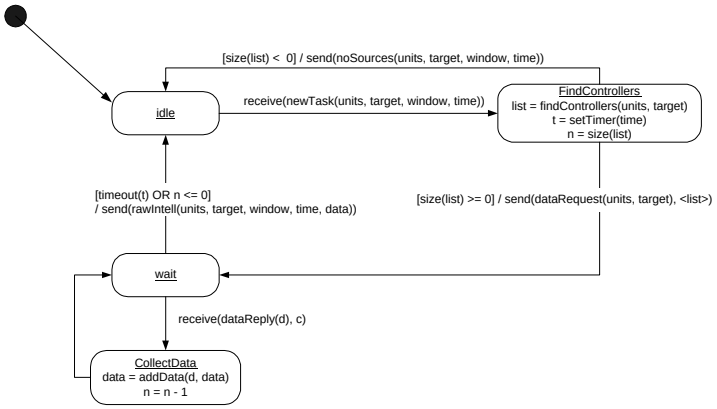

Fig. 6. MaSE Task

2.4 Creating Agent Classes

In the Creating Agent Classes phase of the MaSE methodology, the agent classes are identified from component roles. The product of this phase is an Agent Class Diagram, shown in Figure 7, which depicts agent classes and the conversations between them. The boxes in the figure are the agent classes, containing the class name and its assigned roles. Lines with arrows denote conversations and point from the initiator of the conversation to the responder, with the name of the conversation written either over or next to the arrow.

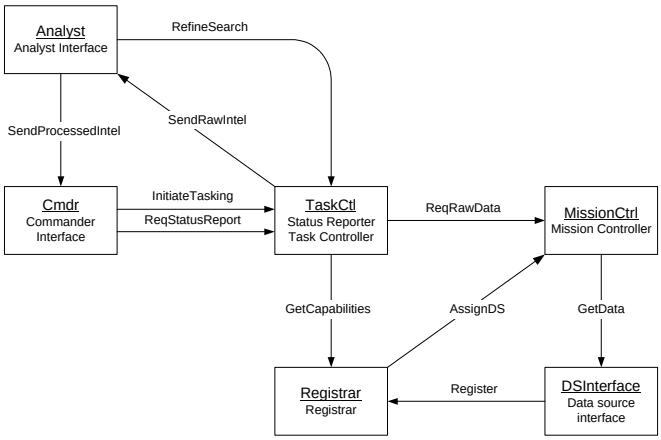

Fig. 7. Agent Class Diagram

During this phase of MaSE, agent classes consist of two components: roles and conversations. In a later MaSE phase, internal details are added to agent classes. The conversations of an agent class are those that it participates in, either as an initiator or responder.

The primary difference between the Agent Class Diagram and similar object diagrams is the semantics of the relationships between agent classes. In Agent Class Diagrams, these relationships define conversations that are held between agent classes. In fact, the primary purpose of this phase is to identify the agent classes that "anchor" each side of a conversation.

Just as before, when mapping goals to roles, there is generally a one-to-one mapping between roles and agent classes. However, the designer may combine multiple roles in a single agent class or map a single role to multiple agent classes. Since agents inherit the communication paths between roles, any paths between two roles become a conversation between their respective classes. As such, it is desirable, where possible, to combine two roles that share a high volume of message traffic. When determining which roles to combine, size and frequency of communications are important, not just the number of communication paths.

2.5 Constructing Conversations

Constructing Conversations is the next phase of MaSE. It is closely linked with the phase that follows it, Assembling Agents. As will be discussed later, it is often beneficial to alternate between the two phases. A MaSE conversation defines a coordination protocol between two agents. Specifically, a conversation consists of two Communication Class Diagrams, one each for the initiator and responder. A Communication Class Diagram is a pair of finite state machines that define the conversation states of the two participant agent classes. The *initiator* side of a conversation is shown in Figure 8 with its associated *responder* side shown in Figure 9. The initiator begins the conversation by sending the first message.

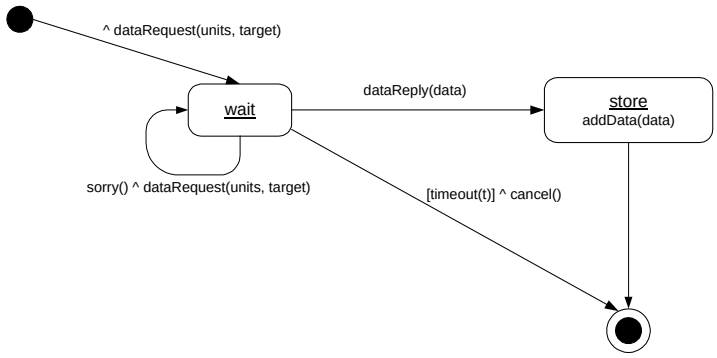

Fig. 8. Initiator Communication Class Diagram

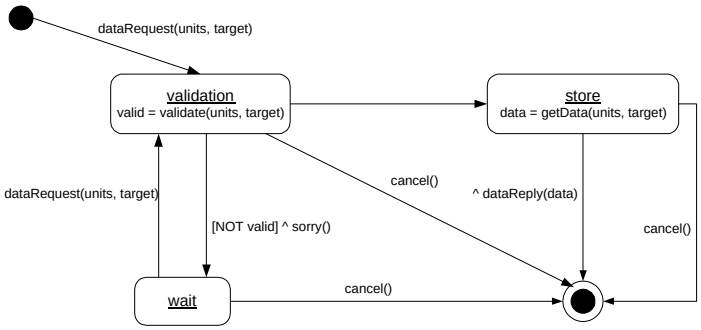

Fig. 9. Responder Communication Class Diagram

When an agent receives a message, it compares it to its active conversations. Upon a match, the agent transitions the appropriate conversation to a new state and performs any required activities from either the transition or the new state. Otherwise, the agent compares the message to all possible conversations that it may participate in with the agent that sent the message, and begins a new conversation if the message matches a transition from the start state. Any activities in a conversation, which may occur in a state or on a transition, are mapped to methods in the corresponding agent classes. The syntax of a transition follows conventional UML notation as shown below, and described in [3].

```
rec-mess(args1)[cond]/activity^trans-mess(args2)
```

While the operation of a conversation is relatively simple, its design can be quite complicated. Conversations are defined at a high level. Specifically, the initiator and responder agent classes are specified for each conversation in the system. The problems encountered in this phase deal with building the finite state automata that define the operation and protocol of conversations.

Conversations must support and be consistent with all sequence diagrams derived earlier. They may also incorporate states from tasks. Some tasks, in fact, operate entirely over single conversations and can be designed directly. In general though, conversations are built by first adding all possible states and transitions that can be derived from the Sequence Diagrams and tasks. At this point, much of the conversation often exists. For the rest of the conversation design, it is a matter of adding states and transitions as necessary to convey the required messages and provide robust operation. Automatic verification of conversation correctness is addressed by Lacey in [13].

2.6 Assembling Agent Classes

In this phase of MaSE, the internals of agent classes are created. Work by Robinson [18] describes the details of assembling agents from a component-based architecture. He defines five different architectural style templates: Belief-Desire-Intention (BDI),

reactive, planning, knowledge based, and a user-defined architecture. Each architecture template has a specific set of components. For example, a reactive architecture includes a Controller, MessageInterface, RuleContainer, and Effectors.

A designer can either define components from scratch or use pre-existing components. Furthermore, components may have sub-architectures containing components. Components are joined with either inner- or outer-agent connectors. Inner-agent connectors (thin arrows) define visibility between components while outer-agent connectors (thick dashed arrows) define connections with external resources such as other agents, sensors and effectors, databases, and data stores. Internal component behavior may be represented by formal operation definitions as well as state-diagrams that represent events passed between components. An example of a component-based architecture is shown in Figure 10.

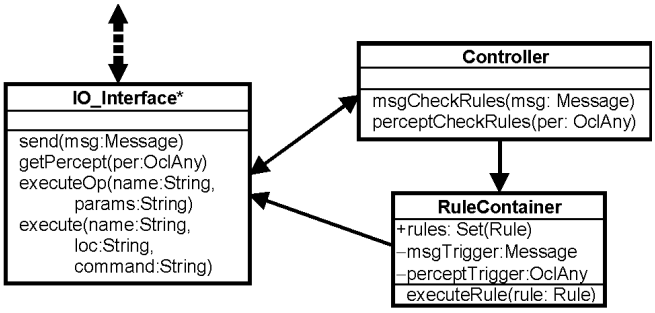

Fig. 10. Generic Reactive Agent Class Architecture

2.7 Constructing Conversations versus Assembling Agent Classes

As discussed in their respective sections, constructing conversations and agent class assembly are closely related activities. In practice, it is useful to alternate between these phases while staying within one functional area of the design. The question of which to do first is answered best by the style of conversations the system uses. In particular, is the system communication-heavy? Are the communications relatively complex? The designer should design conversations first if the system consists of many simple conversations, or if the initial context of the system includes many use cases. It is generally better to define the agents first if there are complex conversations, or if many of the agent classes are being reused.

2.8 System Design

The final phase of the MaSE methodology takes the agent classes and instantiates them as actual agents. It uses a Deployment Diagram to show the numbers, types, and locations of agents within a system. System design is actually the simplest phase of

MaSE, as most of the work was done in previous steps. The idea of instantiating agents from agent classes is the same as instantiating objects from object classes in object-oriented programming.

Deployment Diagrams are used to define a system based on agent classes defined in the previous phases of MaSE. Deployment Diagrams define system parameters such as the actual number, types, and locations of the agents within the system. Figure 11 shows an example Deployment Diagram. The three dimensional boxes are agents, and the connecting lines represent conversations between agents. The agents are named either after their agent class, or in the form of "designator: class" if there are multiple instances of a class. A dashed-line box indicates that agents are housed on the same physical platform.

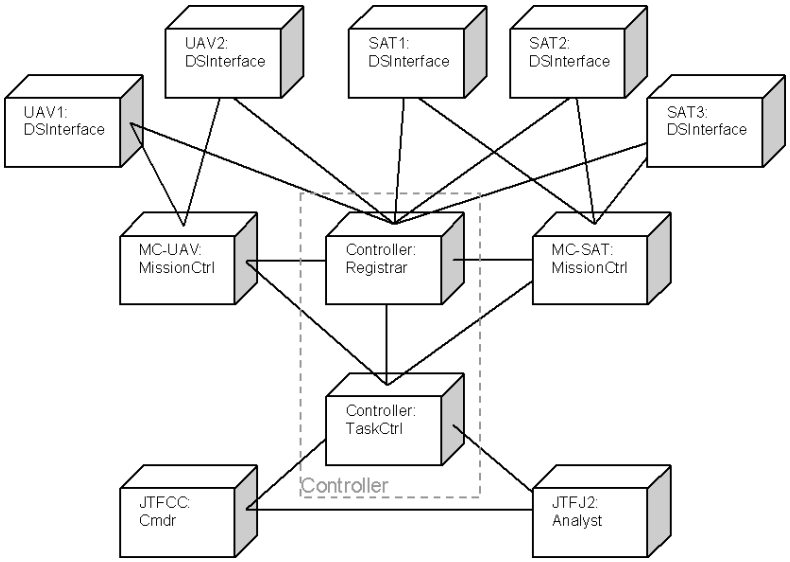

Fig. 11. MaSE Deployment Diagram

A system must be arranged in a Deployment Diagram before it can be implemented in code. This is due to the differences between agents and agent classes. An agent requires information such as a hostname or address to participate in a multiagent system. A Deployment Diagram also offers another opportunity for the designer to tune the system. Agents can be arranged among various machine configurations to take advantage of the available processing power of network bandwidth.

A final element to consider is automatic code generation. The MaSE methodology is concerned with actually engineering agent systems. As such, all of the steps of the methodology work toward that end. It is our vision that code generation be a largely automatic process. Code generation is not a part of MaSE at this time, but is assumed to happen just after this phase.

3. Contributions

MaSE guides a multiagent system designer through the entire software development lifecycle, beginning from a textual system representation and proceeding in a structured manner toward a working implementation. MaSE combines several pre-existing models into a single structured methodology. Most of the models used within the methodology have therefore been already justified and validated within the realm of agents and multiagent systems. A sequence of guided transformations connects the elements of this strong foundation together into a clear high-level picture of how a designer should go about creating a multiagent system.

In conjunction with the MaSE methodology, we have developed a tool, called agentTool, to support the development of multiagent systems using MaSE [5]. The agentTool system currently supports the entire lifecycle from the Goal Hierarchy diagram down to code generation. Developing the methodology and tool together allowed us to focus the methodology toward automation. Focusing on automation forced us to define an unambiguous semantics for the models as well as the relationships between the models. Using MaSE and agentTool we have shown that you can develop a multiagent systems development methodology, along with an automated toolset, that supports multiple types of agent architectures, languages, and communications frameworks.

4. MaSE Applications

MaSE has been successfully applied in numerous graduate-level projects as well as several research projects. The Multi-Agent Distributed Goal Satisfaction project [20] is a collaborative effort between AFIT, the University of Connecticut, and Wright State University where MaSE is being used to design the collaborative agent framework to integrate different constraint satisfaction and planning systems. The Agent-Based Mixed-Initiative Collaboration project [2] is also using MaSE to design a multiagent system focused on distributed human and machine planning. MaSE has been used successfully to design an agent-based heterogeneous database integration system [14] as well as a multi-agent approach to a biologically based computer virus immune system [7].

5. Acknowledgements

This research was supported by the Air Force Office of Scientific Research (99NM097) and the Dayton Area Graduate Studies Institute (HE-WSU-99-09). The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

References

[1] Brazier, F., Jonker C., Treur, J.: Principles of Compositional Multi-Agent System Development. Proceedings of the IFIP'98 Conference (1998).

[2] Cox, M., Kerkez, B., Srinivas, C., Edwin, G., Archer, W.: Toward Agent-Based Mixed-Initiative Interfaces. In Proceedings of the 2000 International Conference on Artificial Intelligence. CSREA Press (2000).

[3] DeLoach, S.A.: Multiagent Systems Engineering: a Methodology and Language for Designing Agent Systems. Proceedings of Agent Oriented Information Systems '99 (1999) 45-57.

[4] DeLoach, S. A., Wood M. F.: Multiagent Systems Engineering: the Analysis Phase. Technical Report, Air Force Institute of Technology, AFIT/EN-TR-00-02, June 2000.

[5] DeLoach, S.A., Wood, M.F.: Developing Multiagent Systems with agentTool. The Seventh International Workshop on Agent Theories, Architectures, and Languages, (2000).

[6] Drogoul, A., and Collinot A.: Applying an Agent Oriented Methodology to the Design of Artificial Organizations: A Case Study in Robotic Soccer. Autonomous Agents and Multi-Agent Systems, 1(1), 113-129.

[7] Harmer, P.K., Lamont, G.B.: An Agent Architecture for a Computer Virus Immune System. Genetic and Evolutionary Computation Conference (2000).

[8] Iglesias, C., Garijo, M., Gonzalez, J.: A Survey of Agent-Oriented Methodologies. In: Müller, J.P., Singh, M.P., Rao, A.S., (Eds.): Intelligent Agents V. Agents Theories, Architectures, and Languages. Lecture Notes in Computer Science, Vol. 1555. Springer-Verlag, Berlin Heidelberg (1998) 185-198.

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

[10] Kendall, Elizabeth A.: Agent Software Engineering with Role Modelling. In this volume (2000).

[11] Kendall, Elizabeth A., and Zhao, L.: Capturing and Structuring Goals. Workshop on Use Case Patterns, Object Oriented Programming Systems Languages and Architectures (1998).

[12] Kinny, D., Georgeff, M., Rao, A.: A Methodology and Modelling Technique for Systems of BDI Agents. Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '96. Lecture Notes in Artificial Intelligence, Vol. 1038. Springer-Verlag, Berlin Heidelberg (1996) 56-71.

[13] Lacey, T., DeLoach, S.A.: Automatic Verification of Multiagent Conversations. Proceedings of the Eleventh Annual Midwest Artificial Intelligence and Cognitive Science Conference, (2000) 93-100.

[14] McDonald, J.T., Talbert, M.L., DeLoach, S.A.: Heterogeneous Database Integration Using Agent Oriented Information Systems. Proceedings of the International Conference on Artificial Intelligence (2000).

[15] Nwana, H. S.: Software Agents: An Overview. Knowledge Engineering Review. 11(3): 205-244 (1996).

[16] Pressman, R.S.: Software Engineering: A Practitioners Approach, 3rd ed. McGraw-Hill Inc., New York (1992).

[17] Raphael, Marc J., DeLoach, S.A.: Marc J. Raphael & Scott A. DeLoach. A Knowledge Base for Knowledge-Based Multiagent System Construction. Proceedings of the National Aerospace and Electronics Conference (2000).

[18] Robinson, D.J.: A Component Based Approach to Agent Specification. MS thesis, AFIT/ENG/00M-22. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base Ohio, USA (2000).

[19] Rumbaugh, J.: Object-Oriented Modeling and Design, Prentice-Hall Inc., Englewood Cliffs, New Jersey (1992).
[20] Saba, G.M., Santos, E.: The Multi-Agent Distributed Goal Satisfaction System. Submitted to International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000).
[21] Sycara, K. P.: Multiagent Systems. AI Magazine 19(2): 79-92 (1998).
[22] Wood, M. F.: Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems. MS thesis, AFIT/GCS/ENG/00M-26. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB Ohio, USA (2000).
[23] Wooldridge, M., and Jennings, N.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review, 10(2): 115-152 (1995).
[24] Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems. 3 (3): (2000).