# Integrating Ontologies into Multiagent Systems Engineering

Jonathan DiLeo[1], Timothy Jacobs[2], Scott DeLoach[3]

[1] College of Aerospace Doctrine, Research and Education, 620 Chennault Circle, Maxwell AFB, AL, USA 36112-6428
Jonathan.Dileo@maxwell.af.mil

[2] Air Force Institute of Technology, AFIT/ENG, 2950 P Street, Building 640, Wright-Patterson AFB, OH, USA, 45433
Timothy.Jacobs@afit.edu

[3] Kansas State University, Department of Computing and Information Sciences, 234 Nichols Hall, Manhattan, KS 66506
sdeloach@cis.ksu.edu

**Abstract.** Multiagent systems have received much attention in recent years due to their advantages in complex, distributed environments. A number of methodologies have been proposed for engineering multiagent systems, however, these methodologies do not adequately address the information domain of the system, which is an integral part of designing proper system execution. Previous work at the Air Force Institute of Technology (AFIT) has developed a methodology for analyzing, designing, and developing multiagent systems, called Multiagent Systems Engineering (MaSE). This research extends the MaSE methodology to include the use of ontologies for information domain specification. The extensions allow the designer to specify information flow by using objects from the ontology as parameters in agent conversations. The developer can then ensure system functionality by verifying that each agent has the information required to accomplish the system goals.

## Introduction

In recent years, interest in multiagent systems has increased as software designers look for new methods of solving problems in complex environments. As agent technology has progressed, research into agent-oriented software engineering has attempted to develop methodologies to help develop robust and reliable multiagent systems [10].

Constructing multiagent systems involves all the problems of traditional distributed systems along with the problems that arise from the behavior of the individual agents. For instance, the individual behavior of the agents can combine to form emergent system behavior that is adverse to proper system execution. Designers need an engineering approach for system development of multiagent systems to address and avoid these problems.

Integrating legacy systems further increases the difficulties of designing multiagent systems due to the varying semantics of different information systems. Different systems can have different terms that represent the same concept, introducing a complication in integrating the systems. Any complete methodology for building multiagent systems must address the data models used by the system and the individual components in the system.

Currently several methodologies exist for building multiagent systems, each varying in their level of development [5,9,15]. MESSAGE [5] is one of the few methodologies that address the information domain of the system. Just as important as the system's representation of the information domain is the various agents' information domain view. Heterogeneous systems can contain agents with differing data models, a case that can occur when reusing previously built agents or integrating legacy components into the system. Most existing methodologies lack specific guidance on the development of the information domain specification for a multiagent system and for the agents in the system.

Failure to incorporate information domain specifications results in designs that do not fully address multiagent system behavior. The interaction between agents in the system frequently occurs through communication. Although communication can occur by the simple passing of performative messages, systems of a complex nature will require the passing of information between agents in the form of parameters. Without specifying the information domain of the system, the designer cannot specify what data types are passed as parameters and cannot describe the information flow in the multiagent system.

The Multiagent Systems Engineering (MaSE) methodology has been developed at the Air Force Institute of Technology (AFIT) to assist in the development of multiagent systems by leading the designer from the initial system specifications to a set of formal design models [3]. The transformations from each step in MaSE are formally defined and provide the engineering approach needed for multiagent system engineering. Despite its benefits in multiagent systems design, MaSE does not currently address the design of the information domain, leading the developer to construct a set of design documents that do not fully specify the semantics of the data passed between agents.

In this paper, we expand MaSE to include ontologies as a mechanism for specifying the information domain of the system and the individual agents. The next section presents an overview of MaSE before ontologies were introduced, followed by a discussion of the changes made to the methodology. The extended MaSE is analyzed in a later section based on experiences of using it to design a distributed course scheduling system.

## Background – Multiagent Systems Engineering

The Multiagent Systems Engineering (MaSE) methodology has been researched at the Air Force Institute of Technology for the last few years. Research focuses on developing a robust methodology for constructing multiagent systems. MaSE divides the development of multiagent systems into analysis, design, and implementation

phases [3]. MaSE originally consisted of three steps in the analysis phase and four steps in the design phase as shown in Figure 1. The developers of MaSE intended for these phases and steps to be applied iteratively. During system implementation the models from the analysis and design phases are used to program the system into code. This paper discusses the addition of the System Ontology step, shown in Figure 1, into the MaSE methodology.

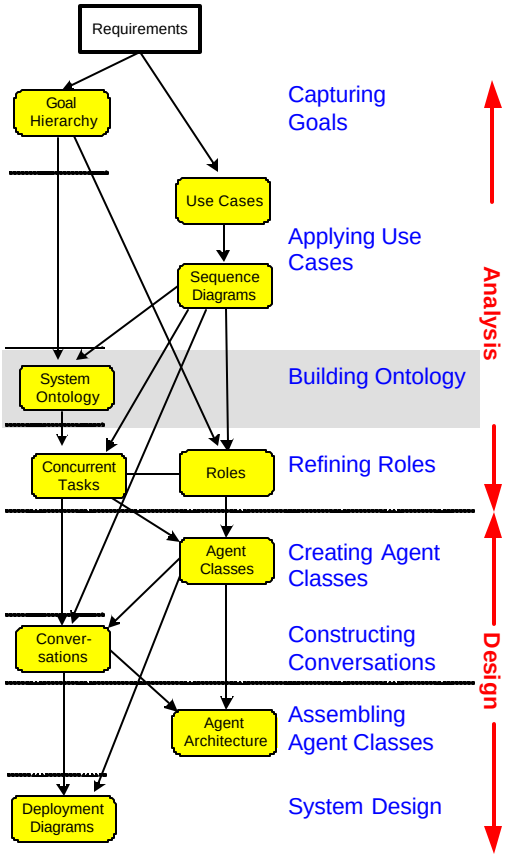


**Fig. 1.** Extended MaSE Phases, Steps and Models

**Analysis Phase**

The Analysis Phase is concerned with establishing a set of roles and assigning tasks to those roles to describe the system requirements. The Capturing Goals step starts this phase with the transformation of the initial system specification into a goal hierarchy.

This step allows the designer to organize the goals that the system needs to accomplish.

The basic scenarios that the system should perform are captured in the Applying Use Cases step. Use cases are created and then transformed into sequence diagrams. The final step of analysis, Refining Roles, uses the outputs from the previous two steps to create roles and assign the tasks to be performed by those roles in the system. Tasks are associated with each role to describe the behavior that the role must have to accomplish its assigned goals. Figure 2 is an example of a Role Model in MaSE, that indicates the goals each role is assigned, the tasks associated with each role, and the communication between the tasks using the protocols indicated by arrows.

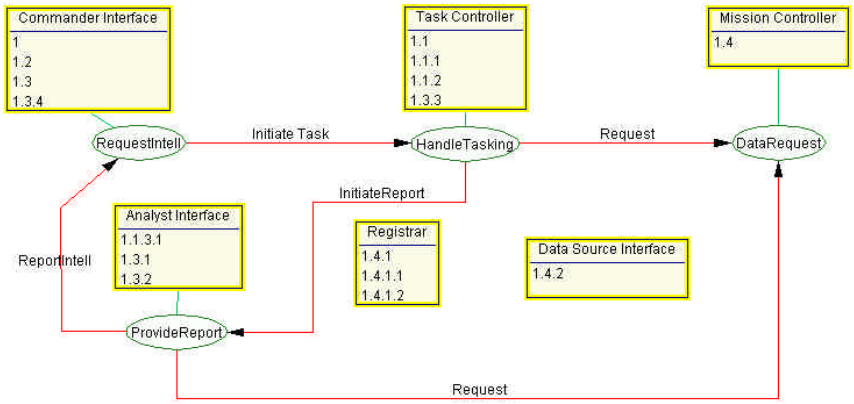


**Fig. 2.** Example MaSE Role Model [2]

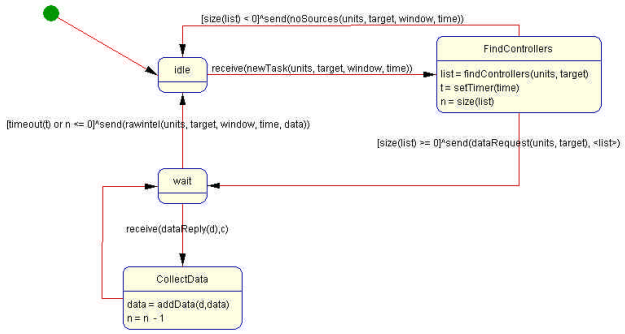


**Fig. 3.** HandleTasking Concurrent Task Diagram [2]

Tasks are defined graphically using a finite state automaton, as shown in the Concurrent Task Diagram of Figure 3. Transitions in the concurrent task diagrams

follow the syntax *trigger [guard] ^ transmission(s)*. This syntax represents that a transition is enabled only when the event *trigger* occurs and the condition *guard* evaluates to true. When the transition is executed, the specified *transmission(s)* will be executed by the role. Once in a state, the task remains in that state until all actions defined in that state have been completed and a transition out of the state has been enabled.

**Design Phase**

Once the requirements for the system have been modeled during the Analysis Phase, the design of the multiagent system begins. The first step in the Design Phase is Creating Agent Classes, where the roles are assigned to specific agent classes. This step creates an Agent Class Diagram, as shown in Figure 4, that shows the classes in the system, the roles played by the agent classes (listed under the agent class name), and the conversations between classes (denoted by arrows pointing from the initiator class to the responder class).

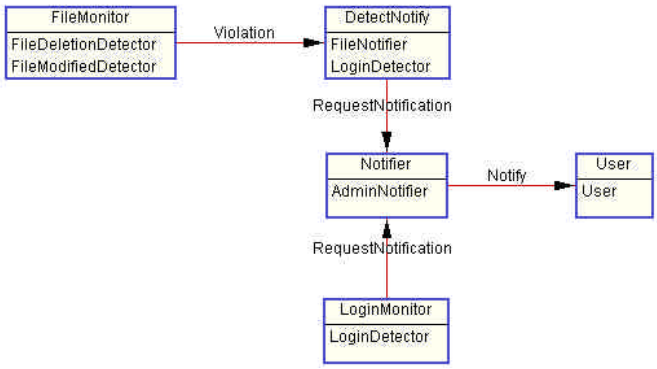


**Fig. 4.** Example Agent Class Diagram [3]

Details of the conversations are defined in the Constructing Conversations step, where finite state automata are used to show the states in a conversation. Each conversation has two diagrams: one for the initiator and one for the responder of the conversation, with Figure 5 demonstrating the diagram for the initiator of the *RequestNotification* conversation of Figure 4. The set of conversations that an agent class participates in is derived from the communications of the roles that the agent plays. For example, the *FileNotifier* and *AdminNotifier* roles are defined by concurrent tasks that communicate with each other. Since these two tasks are in separate agent classes in Figure 4, this communication becomes the *RequestNotification* conversation.

The third step in design, Assembling Agent Classes, defines the components of the agent architecture, allowing for the logical decomposition of agents. The final step of System Design creates a Deployment Diagram to show the amount and location of

each type of agent in the system. The outputs from the design steps describe the actions and conversations used in the multiagent systems. The semantics of the parameters passed in those conversations were not previously defined in the MaSE process.

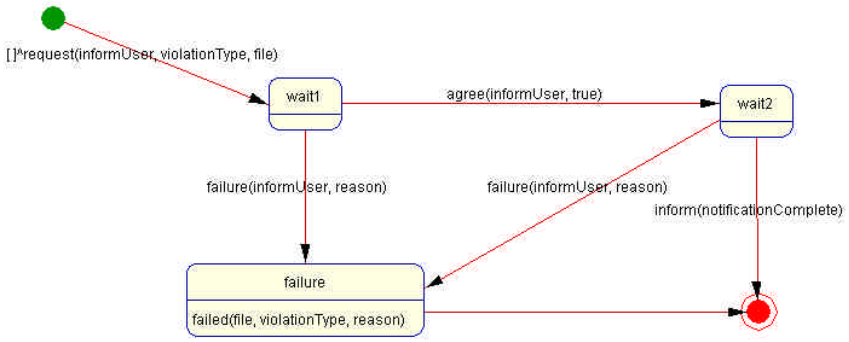


**Fig. 5.** RequestNotification Conversation Diagram [3]

**agentTool**

agentTool is an automated development environment that supports the MaSE methodology by allowing the designer to create the MaSE Models in a graphical environment [3]. Using transformations defined in [13], agentTool allows for the automatic generation of design documents based on the role models and task diagrams. Once the designer specifies the system in agentTool, the program can output Java code for the system. Because information domain specifications are not incorporated in the MaSE methodology, the code produced from agentTool classifies all parameters as Java Objects. The implementer must then go through and change the declarations to the appropriate data structure used to represent the semantic concepts of the parameters. Including the construction of the system data model removes the necessity of this step, as the parameters can automatically be specified as the correct type.

## Including Ontologies in Multiagent Systems Engineering

The word *ontology* was taken from philosophy where it represents the study of the nature of being. Much debate exists on the exact definition of an ontology when used for knowledge engineering or artificial intelligence [8]. The most common definitions state that an ontology is a specification of a conceptualization [7] or that an ontology is the shared understanding of some domain of interest [14]. This research uses the latter definition, specifically that an ontology defines classes,

functions, object constants, and axioms to constrain meaning of some type of world view of a given domain.

The classes in an ontology describe the objects in the system's view of the information domain. The functions and object constants define the properties of and relationships between those objects. The analyst uses axioms to describe any additional relationships or restrictions on concepts in the ontology. For instance, an axiom might specify that a student must be registered or signed up to take at least one class. With these concepts, the ontology describes the concepts and relationships used to interact in the domain.

In this research, we use ontologies to specify the information domain of a multiagent system. Just as it is important to specify the data model in a traditional software development process, the data model for a multiagent system must also be specified. The agents in the system interact by passing messages and these messages frequently involve passing parameters. These parameters are objects of some sort, and without an information domain specification, the methodology cannot address the information contained in these parameters.

To use ontologies to describe the information domain of a system in MaSE, this research introduces a new step in which the designer constructs the ontology for the system during the MaSE analysis phase. We determined the step should occur after the Applying Use Cases step. This placement allows the designer to use terms from the goal hierarchy, use cases, and sequence diagrams as possible concepts in the ontology. The resultant ontology can be used to create tasks in the refining roles step of MaSE. Tasks often indicate parameter passing, so the step is placed after the construction of the ontology to allow the designer to specify the type of the parameters as classes from the ontology. The extended MaSE diagram is shown in Figure 1.

**Constructing Ontologies for Multiagent Systems**

An appropriate methodology for developing ontologies must be defined for designers to use for specifying domain representations in multiagent systems. The existing methodologies for designing domain ontologies are built to describe everything about a specific domain; however, this is not appropriate for multiagent systems because the system ontology should only specify the information required for proper system execution. The system ontology acts as a prerequisite for future reuse of the system, as the ontology specifies the view of the information domain used by the multiagent system. Any system that reuses the developed multiagent system must ensure that the previously developed system ontology does not conflict with the ontology being used in the new system.

Reinventing the wheel, by developing a whole new methodology, does not make sense, because many years of research have gone into developing domain ontology methodologies. Instead, we extract the main steps common to the IDEF5 and Methontology methodologies [11, 6].

To construct the ontology, the designer first determines the purpose and scope of the ontology and then collects and analyzes data from the information domain for possible use in the ontology. Finally, the analyst constructs the initial ontology and

refines, validates, and matures the model into a complete ontology. Each of these steps is discussed in detail below.

**Define Purpose and Scope of Ontology**

The designer must describe why the ontology is being developed as well as the range of intended users of the ontology. This facilitates ontology reuse by allowing others to quickly see the reason the ontology was constructed and what information the ontology contains. For example, when designing a multiagent system to perform course scheduling, the ontology must define classes regarding courses, quarters, instructors, classrooms, etc. The software requirements and the goal hierarchy help define the purpose of the ontology, as the purpose of the ontology is to fulfill the information needs of the multiagent system. The purpose describes why the ontology exists, such *as to list all classes in the education domain required when scheduling courses*. The scope defines the level of detail to which the ontology describes the objects, such as *defining only the semantic ideas necessary to schedule courses in a distributed network environment.*

To further define the scope, the designer can utilize the previously identified use cases to determine the types of data that the system will use. For example, a use case may describe one agent passing another agent a specific course to schedule. The designer uses this situation to determine the level of detail necessary to describe a course so that the system can properly execute the events described in the use case.

**Collect Data**

Having defined the scope, the analyst knows the level of detail and domain the ontology represents and can start building the model. The designer first creates a list of possible terms or concepts that the ontology must describe. Designers form this list by examining the goal hierarchy, use cases, and sequence diagrams from the previous MaSE steps for candidate ontology terms. The analyst looks for nouns in these models that could represent some type of information in the system. For instance, when reviewing the sequence diagram for scheduling courses, the analyst might place *schedule*, *lock*, *section*, *student*, *instructor*, and *classroom* in the candidate list, if these terms appear in the sequence diagram. The actions in the sequence diagram illustrate that these terms could be part of the information passed in the system. The designer examines the system requirements, goal hierarchy, and use case models in a similar manner to create the candidate term list for the ontology.

**Construct Initial Ontology**

To construct the initial ontology, the designer takes the list of terms or concepts and organizes them into classes and attributes and produces an initial draft of the data model. When creating the ontology, the analyst must remember to only specify the concepts that the system needs to accomplish its goals. For example, the ontology should not specify all attributes of a *Human*, such as *height*, *age* and *weight*, when the system only requires the name of a *Human* to function.

Before creating an entirely new ontology, the designer must determine whether any existing ontologies can meet the system needs. The user reviews ontology libraries and existing company data models looking for objects that resemble the concepts

listed in the term list built earlier. The benefit to using an existing ontology is that the system is interoperable, in terms of passing data, with any other system that uses the same data model. If no existing ontologies fully specify the information needed for the system, the designer must build a new ontology. If the designer finds an ontology that partially satisfies the system needs, that ontology can be used as a starting point for the new ontology. Users should post created models in some shared repository so that others can reuse the data model, increasing the interoperability of future systems.

Continuing the course-scheduling example, the analyst decides what terms from the candidate list should become classes and then organizes them into the class hierarchy shown in the left most pane of Figure 6 Terms that will become attributes of the classes or that the designer decides are not necessary as classes are not included. For instance, the *lock* term identified earlier is not included in the example ontology. The analyst can decide that the term will be implemented in a manner that will not require the creation of a *Lock* class. The attributes of each class describe the properties of the class and the relationships to other classes. Figure 6 shows how the *Schedule* class has two attributes: *composedOfSections* and *courseTypes*. The *composedOfSections* attribute is multiple, indicating it is a list of the *Section* objects included in the schedule. The *courseTypes* attribute is a list of String objects used to describe the type of courses that the schedule contains, such as AERO or CSCE.

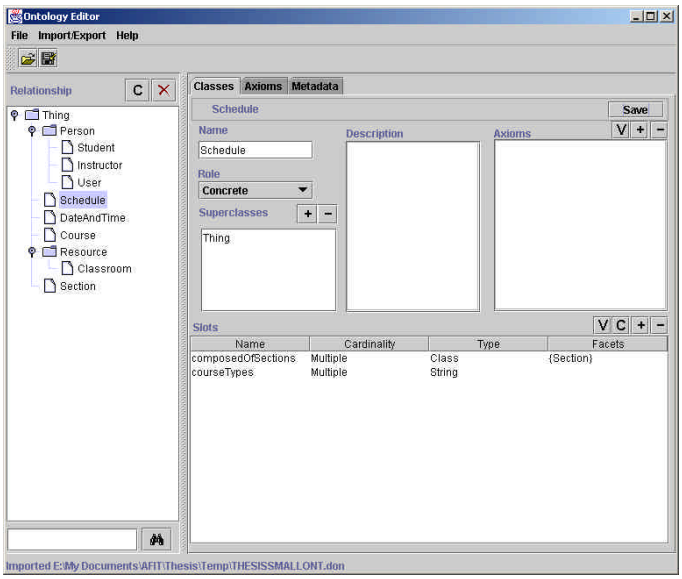


**Fig. 6.** Ontology Editor Program

**Refine and Validate Ontology**

Once the designer defines the classes, attributes, and axioms of the ontology, he must validate that the ontology meets the system requirements. To validate the model, the analyst examines the situations described in the use cases and sequence diagrams to

ensure that the ontology describes all the information needed in the execution of those scenarios. Any missing information is added to the ontology, and any extraneous information is removed from the ontology. If any information is incorrectly specified, the designer makes the necessary corrections to the ontology.

This step is repeated throughout the development of the system. If at any time the designer realizes that some piece of information is missing from the ontology, the ontology is modified to include this information. The ontological construction is complete once the analyst is satisfied that the ontology represents all the necessary information from the sequence diagrams and use cases.

In the course-scheduling example, the analyst realizes there is a problem when trying to schedule the classes in the use cases. Instructors and students might have other events that conflict with the scheduling of sections, such as department meetings, but the ontology is incapable of representing these events. The project adds a *ScheduledEvent* object as a parent of a *Section* to represent these other events. A *Person* then has a *scheduledEvents* attribute that contains a list of all events that individual is required to attend. With these additions, the scheduler can verify that an individual has no activity scheduled during a specific time frame.

**Using Ontologies in Multiagent Systems**

Once the system ontology is constructed, a multiagent system design methodology should allow the analyst to specify objects from the data model as parameters in the conversations between the agents. To ensure the proper functionality of the multiagent system, the designer must be able to verify that the agents have the necessary information required for system execution. Since the information is represented in the classes of the data model, the design of the methodology must show the classes passed between agents. We modified the Refining Roles and Constructing Conversations steps that involve message passing, to include specification of the types for the parameters passed in the messages in MaSE.

Previously, parameters in the Conversation State and Concurrent Task diagrams were described only by their name. Now, the data type of each parameter is specified along with their name. For example, in Figure 7, the analyst specified that the parameter *schd* is of a type *Schedule*, that is defined in the ontology. Actions can now use the information contained in the attributes of the parameters, as the parameter type is known along with the attributes of that data type. The internal variables of the tasks and conversations can also be typed according to the system ontology. Using the data values of the parameters and variables, the validity of the conversations can be automatically verified. This benefit is discussed in the next section.

Along with building a system data model, the multiagent system design methodology should allow agents to have their own individual data models. By addressing this capability, the methodology allows for the development of heterogeneous systems. The requirement for a multiagent system to integrate with existing systems often creates such heterogeneous systems. With the various data models comes the requirement to show how the information models relate.

The methodology should provide the ability for the designer to show the relations between the data models in some manner. Showing the relationships indicates to the

code developers what information from one model is required to create objects in the other model. Without describing these relations, the developers may not be able to know how to translate objects in a conversation between two agents with separate data models, as each agent uses different classes to describe the information.

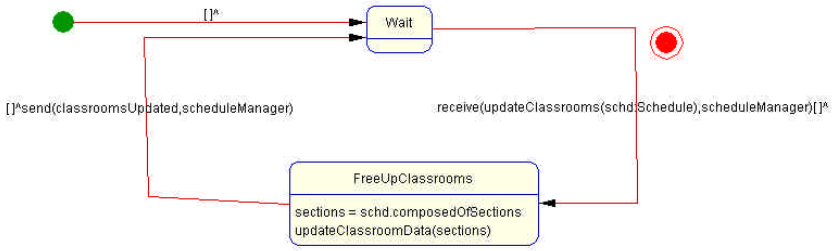


**Fig. 7.** Sample Finite State Automata with Information Passing

The Assembling Agent Classes step now includes the ability for designers to specify ontologies for the different agent components and to map them to the overall system ontology. These component ontologies can be used to represent the data models of existing agents or systems that are included in the system. The designer could map the component ontologies to one another or two the system ontology. When mapping between the component ontologies, if one component interacts with many different components that each have a separate ontology, a large number of mappings will result. If there are $n$ agent components, and each component interacts with every other component, there will be $(n-1)!$ mappings. By showing the component ontologies relationships to the system ontology, the system will at have at most $n$ mappings, one for each component. The designer maps to the system ontology to reduce the number of possible mappings required.

To describe the relationships between the ontologies, the analyst specifies which objects and attributes correspond in the ontologies. For instance, Figure 8 shows that the *Course* object from the component ontology matches the *Class* object in the system ontology. It also shows that the *Course Number* attribute of a *Course* corresponds to the *number* attribute of a *Class*. With these relationships defined, an implementer

To assist the user with the ontological mapping, we developed an information retrieval ranking model that computes the similarity between ontology classes. Typically in information retrieval, the user specifies a query and the search engine ranks the documents in a collection based on their similarity to the query. We consider the mapping process similar to an information retrieval process, where the user is searching for a class in the system ontology that is similar to the class in the component ontology.

The ranking model uses the characteristics and attribute structures of two classes to compute their similarity score. In this manner, the user can be presented with a list of classes from the system ontology ranked by their similarity to a given class in the component ontology. For example, in , the ranking model returned the *Class* object

as the third most similar to the *Course* object. The attribute structures of the *Section* and *Student* objects resembled the structure of the Class object, and were thus returned as more similar.

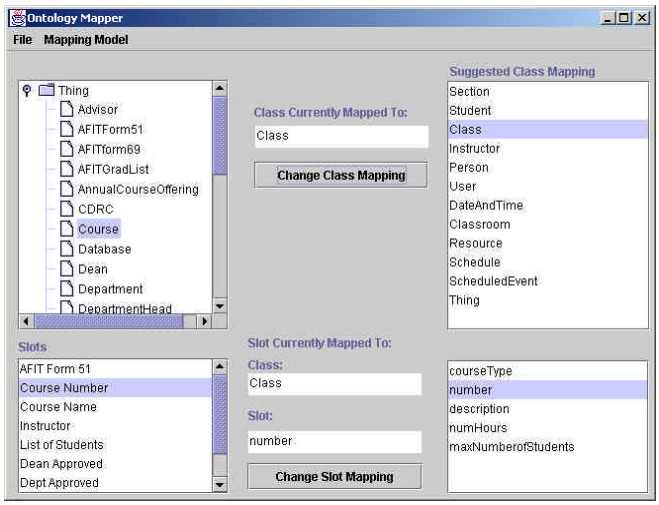


**Fig. 8.** Ontology Mapper Program

## Analysis

To test the ability of MaSE to assist the user in developing the system and agent ontologies and to evaluate the benefits of including the information domain in the development of multiagent systems, we designed a distributed course scheduling system using the extended MaSE [4]. The rest of this section discusses the experiences of using MaSE based on the design of the sample system.

The creation of the system ontology occurs at the appropriate time in the development process. The steps in MaSE prior to constructing the ontology, Capturing Goals and Applying Use Cases, provide a set of terms for consideration as possible objects in the ontology. The step after the ontology creation passes messages and information amongst the roles. This information passing occurs with parameters that are specified as objects from the ontology. By placing ontology creation right before task creation, the methodology allows the user to analyze the problem domain thoroughly before creating the data model. Designers can determine exactly what information is necessary for the system while creating the data model before it is required for the rest of the development process.

MaSE describes tasks and conversations using finite state automata, so the experience with using the ontology is the same for each case. With the existing data model, the designer can specify the type of objects passed between agents, as shown in Figure 7. The designer specifies that a *Schedule* object is received as a parameter

in the *updateClassrooms* message. Additionally, the analyst describes the actions that the task performs on the *composedOfSections* attribute of the *Schedule* object to update the classroom information. The data types are now defined, including their attributes, as a part of the development process and can be used in behavior diagrams.

With the ability to specify parameter types, designers also know the exact structure of the information passed between agents and can verify that each agent has all the information needed to accomplish its goals. Previously, the designer passed in a parameter assuming the developers would realize to code that information into that object. With a specified data model, the user can specify the object and ensure the information is encoded as an attribute of the object.

By specifying agent component ontologies, the developer can reuse agents from previous systems or integrate existing legacy systems to form a heterogeneous system. The designer can specify mappings between the component and system ontology to illustrate which data types represent the same semantic concept in the various ontologies.

Another result from the inclusion of ontologies is that agentTool can output more complete code. A Java class is created for each object in the ontology with the appropriately defined attributes. The code for the agents and conversations can then use these classes based on the data types defined in the design diagrams, requiring the developer to modify less code.

The automatic conversation verification mechanisms in agentTool [12] can also be expanded to check the data types of the parameters in conversations. The verification procedure checks for deadlock, non-progress loops, and for valid message sequences comparing the signatures of messages. These signatures can be expanded to include the data types of parameters passed in the messages. This additional checking verifies that the conversations are valid in terms of the data types of the parameters in a sent message matching the parameters of a message received by some agent in the system.

The addition of ontologies also allows for the creation of security related verification in agentTool. The ontology can be expanded to support the classification of the data objects. For example, an *AirTaskingOrder* object defined as a classified. Objects can be specified as classified or unclassified, or the ontology can be expanded to support additional levels of classification. The system designer can then define the classification level of the individual agent classes. The conversations could then be automatically verified to ensure that the system ensures that classified information is not passed to non-classified agents.

The inclusion of ontologies also assists with the development of organizational rules in MaSE. Organizational rules can define constraints between roles and protocols, roles and data, or between roles themselves. The axiom and attribute characteristics in the ontology can define organizational constraints placed on the information. Using the ontology to develop functions that describe the data in the system can expand these constraints. For example, the function *classesRegistered(s)* could return the *Section* objects that the *Student* object *s* is currently registered for. These functions can be combined with protocol functions to describe relationships that organization of agents must ensure are met during system execution [1].

## Conclusions

In this paper, we have shown how we extended MaSE to include the development and use of ontologies to define the information domain of the system. This addition to MaSE allows the user to define a system ontology and to specify data model objects in MaSE behavioral models. The extensions mesh with the previous version of MaSE to ensure the MaSE steps logically flow through a software development process where the outputs of one step become inputs to the following steps. Designers may now construct system and component ontologies by creating and structuring classes and attributes using terms extracted from the goal hierarchy, system requirements, and use cases. Agents can then use these classes to share information with one another.

By adding these steps to address the information domain, this research matures MaSE towards a more complete methodology for building multiagent systems. MaSE now addresses the system's behavioral, structural, and data models, thus defining the aspects required to ensure a coded system will fulfill the initial requirements. With these models, the designer can ensure that each agent has the required information to fulfill all of the system requirements.

When automated in agentTool, the inclusion of ontologies can increase the ability to automatically verify the conversations in the system by comparing the signature of messages. With ontologies, agentTool could also verify the security of the multiagent system, ensuring that classified information is not passed to unclassified agents.

The methodologies built to construct and use ontologies discussed in this paper can be integrated into other agent oriented design methodologies in a manner consistent with the models of each methodology. This paper presents the integration of the concepts into MaSE as an example implementation of including ontological engineering into multiagent systems development methodologies.

## Acknowledgements

This work is sponsored in part by the Information Directorate of the Air Force Research Laboratories in Rome, New York. The views expressed in this paper 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 U.S. Government.

## References

1. DeLoach, S. A.: Modeling Organizational Rules in the Multiagent Systems Engineering Methodology. Proceedings of the 15th Canadian Conference on Artificial Intelligence. Calgary, Canada (2002)
2. DeLoach, S., Wood, M.: Multiagent Systems Engineering: the Analysis Phase. Technical Report, Air Force Institute of Technology, AFIT|EN-TR-00-02 (2000)
3. DeLoach, S., Wood, M., and Sparkman, C.: Multiagent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, Vol 11 no. 3, (2001)

4. DiLeo, J.: Ontological Engineering and Mapping in Multiagent Systems. MS thesis, AFIT/GCS/ENG/02M-03. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH (2002)
5. Evans, R., Kearny, P., Stark, J., Caire, G., Garijo, F., Gomez Sanz, J., Leal, F., Chainho, P., Massonet, P.: MESSAGE: Methodology for Engineering Systems of Software Agents. EURESCOM Project P907. (2001)
6. Fernández, M., Gómez-Pérez, A., Juristo, N.: METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Ontological Engineering: Papers from the 1997 AAAI Spring Symposium. Technical Report SS-97-06, AAAI Press (1997)
7. Gruber, T. A translation approach to portable ontologies. Knowledge 7.
8. Gruninger, M., Lee, J.: Introduction to SPECIAL ISSUE: Ontology applications and design. Communications of the ACM, ACM Press, New York. Vol 45 no. 2 (2002)
9. Iglesais, C., Garijo, M., Gonzalez, J., Velasco, J.: Analyasis and Design of Multiagent Systems using MAS-CommonKADS. INTELLIGENT AGENTS IV: Agent Theories, Architectures, and Languages, Springer Verlag, Berlin Heidelberg (1998)
10. Jennings, N.: On Agent-based Software Engineering. Artificial Intelligence: 117 (2000) 277-296
11. KBSI: The IDEF5 Method Report. KBSI Report, Texas (1994)
12. Lacey, T, DeLoach, S.: Automatic Verification of Multiagent Conversations," in Proceedings of the Eleventh Annual Midwest Artificial Intelligence and Cognitive Science Conference. Fayetteville, Arkansas, AAAI Press (2000) 93-100
13. Sparkman, C., DeLoach, S., Self, A.: Automated Derivation of Complex Agent Architectures from Analysis Specifications. Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001), Montreal, Canada, (2001)
14. Uschold, M., Gruninger, M. ONTOLOGIES: Principles, Methods and Applications. Knowledge Engineering Review. Vol 11 No. 2 (1996)
15. Wooldridge, M., Jennings, N., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems. 3 (3) (2000) 285-312