Single-Model Method for Specifying Multi-Agent Systems

Arnon Sturm
Technion - Israel Institute of Technology
Haifa 32000, ISRAEL
sturm@tx.technion.ac.il

Dov Dori
Technion - Israel Institute of Technology
Haifa 32000, ISRAEL
dori@ie.technion.ac.il

Onn Shehory
IBM Haifa Research Lab, Haifa University
Haifa 31905, ISRAEL
onn@il.ibm.com

ABSTRACT
Multiple modeling methods for constructing agent-based systems have been suggested, however none of them has been accepted as a standard. A prominent reason for this is the gap that exists between agent-oriented methods and the modeling needs of agent-based systems. This gap is, in large part, due to lack of an agreed-upon set of building blocks for modeling Multi-Agent Systems (MAS) and standalone agents, and lack of support for essential software engineering properties. To bridge the gap, we suggest a set of building blocks and an agent-based modeling method. The building blocks should be useful as a basis for developing modeling methods for MAS, and as a benchmark for comparison between such methods. Our proposed modeling method, which is based on the building blocks, is novel in its ability to capture the different aspects of MAS in a single unifying framework. It further excels in providing accessibility, expressiveness and flexibility, which are major lacking software engineering properties in other methods. We demonstrate the usage of the method for modeling MAS, optionally, in conjunction with an existing MAS infrastructure. Thus, our method enhances both the utilization of existing infrastructure and the development of agent-oriented models.

Categories and Subject Descriptors
[D.2.2] Design Tools and Techniques, [D.2.13] Reusable Software

General Terms
Design, Languages

Keywords
Agent-oriented software engineering, Object-Process Methodology, Agent-oriented modeling

1. INTRODUCTION
In recent years, researchers and practitioners have recognized the advantages of applying the agent paradigm for system development. Yet, the number of deployed commercial agent-based applications is quite small. A major reason for this slow technology transfer is the lack of an industry-standard methodology for agent-based application development [8],[9], analogous to UML in the domain of software systems development. To meet this challenge, a new domain of Agent-Oriented Software Engineering (AOSE) is emerging. AOSE supports the introduction of agent-based systems to the industry as an engineering approach [8].

The software engineering (SE) research community widely agrees that modeling is an essential element of system development. Modeling enables planning of the desired system, it can serve as a basis for communication between the system developers and the business customers, it is (usually) easier to understand than code is, and it provides a basis for high-quality documentation. Recognizing these advantages, many MAS[1] development methods that follow SE principles have been developed. Among the known agent-oriented modeling methods are GAIA [24], DESIRE [1], ADEPT [10], FAF [11], MaSE [3], MAS-CommonKADS [7] and AUML [14].

In spite of the proliferation of MAS modeling approaches, none of the AOSE methods has been accepted as a standard, neither is any one of them broadly used. Moreover, neither an AOSE standard nor a common ontology[2] have been devised. Research that examined and compared the properties of existing methods [21] has suggested that none of these methods is fully suitable for industrial development of MAS. Although those methods are based on strong logical and agent-oriented foundations, they lack support for essential software engineering issues, notably accessibility and expressiveness. This lack has had an adverse effect on industry acceptability and adoption of agent technology, as confirmed by [23] as well.

To overcome these problems, this work defines a set of core building blocks for the MAS domain and provides a support framework for developing MAS applications that also caters to a number of critical software engineering aspects. Defining the MAS core building blocks requires understanding of, and agreement upon, these building blocks. Such agreement enhances coherence among agent-based systems and cross-fertilization among their developers and researchers. The proposed set of MAS core building blocks may serve as a basis for an agreement among the agent research community, leading to coordinated work on modeling and implementation issues. This pre-defined set can also be used as a benchmark for comparison [22] among modeling methods for MAS.

To address the lack of SE properties in MAS modeling methods, one may either develop a new MAS modeling method or select an existing modeling method and enhance it for modeling MAS. We have chosen the second alternative, because it enables us to capitalize on proven advantages of the adopted method and

AAMAS '03, July 14-18, 2003, Melbourne, Australia.

[1] When using the term MAS we refer to single agents as well.

[2] Using its SE interpretation, the term ontology in this paper refers to the terminology and building blocks used within a specific domain.

significantly reduce the development effort. The question is, which method should be adopted.

We can possibly adopt a method based on a knowledge engineering approach. However, such methods suffer from problems in modeling and reuse [8] and from limited accessibility and expressiveness [21]. These limitations prohibit the use of such methods as a basis for the sought method, as addressing the particular software engineering aspects of interest will be too complicated.

The main software engineering approaches to system specification are the data-flow approach, the entity-relationship approach and, most dominant, the object-oriented approach. Due to its dominance, we decided to examine the suitability of the object-oriented paradigm as a specification approach for the MAS domain. This has led us to start off with an existing object-oriented method as a candidate for a basis of our method. UML is a leading candidate for becoming a basis for a MAS development method, and has indeed been adopted by AUML, is too complex for the task at hand [19]. In addition, the UML consistency and integrity are questionable as discussed in [6]. While AUML enhances the expressiveness of UML for specifying MAS, it does not address the accessibility aspect, i.e., the ease of use of the method. Alternatively, one can use the Object-Process Methodology (OPM) [4], which inherits its capabilities from both object- and process-oriented paradigms. OPM is an integrated approach to the study and development of software systems. Using a single graphical model and a corresponding automatically generated textual specification, OPM unifies the system's function, structure, and behavior within one frame of reference that is expressed both diagrammatically and by a subset of English.

Probing further, it soon appeared that at least with respect to accessibility and expressiveness, which are the two major aspects our method aims to address, OPM is better than the object-oriented paradigm [17]. In addition, we evaluated OPM using the features-based method of [21], and found that OPM is well-suited for developing MAS applications. OPM is highly expressive, but due to its generic nature, expressing building blocks of a specific domain, such as MAS, may prove to be tedious. To overcome this, we endow OPM with the capability of defining MAS components in a straightforward, user-friendly manner. We call this extension OPM/MAS.

We introduce OPM/MAS as a method for modeling MAS. OPM/MAS follows the suggested MAS building blocks and the specific modeling needs of MAS. Capitalizing on the conciseness and expressiveness of OPM [17], our method overcomes limitations of existing agent-based methods and addresses the accessibility and expressiveness problems that characterize MAS modeling methods.

The contribution of this paper is twofold: First, it introduces a set of MAS candidate building blocks. Second, it introduces a new method for modeling MAS, which addresses major software engineering problems and is based on a single, unified model. The paper is organized as follows. In Section 2 we introduce and define the MAS building blocks. In Section 3 we present OPM and its MAS extension. Section 4 presents a case study that demonstrate the use of this extension. In Section 5 we discuss how OPM/MAS can be used for modeling a specific MAS which is based on an existing infrastructure, and Section 6 concludes with a discussion.

2. MAS BUILDING BLOCKS

Although AOSE exists for several years, there is no formal definition or an agreement on the set of elements that are essential for constructing MAS. Further, no ontological foundation for these has been suggested. This ontological foundation can be presented as a set of building blocks. This set can serve as a basis for better understanding of the MAS domain. In this section we propose a set of building blocks that are based on current research, systems, methodologies and architectures, such as, "Roadmap to agent research and development" [9], "From AOSE methodology to agent implementation" [12], and "GAIA" [24]. These building blocks are listed below along with their definitions[3] and are grouped as follows.

Knowledge and Information– this group of building blocks describes the internal knowledge structure of an agent.

- **Object:** A data container.
- **Ontology Term:** A term that an agent uses for understanding other entities, such as agents, web sites and other systems. The collection of ontology terms is an agent's dictionary. The dictionary enables the agent to understand messages which are not explicitly correlated with its internal structure.
- **Fact:** A knowledge item about the domain of discourse. The state of the world can be represented as a set of facts.

Specializations of a fact are a Belief, a Desire, and an Intention. These provide the building blocks for BDI–based [19] systems, which are widely used.

- **Belief**: A fact that is believed to be true about the world.
- **Desire:** A fact of which the current value is false and the agent (that owns the desire) would prefer that it be true. Desires within an entity may be contradictory. A widely used specialization of a desire is a goal. The set of goals within an agent should be consistent.
- **Intention:** A fact that represents the way of realizing a desire.

Behavior – this group of building blocks describes the behavioral aspect of an agent and the communication aspects between agents.

- **Task**: A piece of work that can be assigned to an agent or performed by it. It may be a function to be performed and may have time constraints.
- **Service:** An interface that is supplied by an agent to the external world. It is a set of tasks that together offer some functional operation. A service may consist of other services.
- **Conversation:** An open channel, through which exactly two entities can talk.
- **Message:** A means of exchanging facts or objects between entities.
- **Messaging:** A set of methods by which messages are built and transferred. On the sending end it consists of building the message, associating objects with it, and sending it out. On the receiving end, it consists of

[3] We believe that some researchers may disagree with some of the definitions. Even in the case of disagreement, these definitions can serve as a reference set and a basis for future extensions and improvements.

receiving the message, translating it, and associating it with the appropriate objects.

- **Protocol:** An ordered set of messages that together define the admissible patterns of a particular type of interaction between entities.

MAS architecture – this group of building blocks describes the overall structure of a MAS application. This structure consists of both logical and physical aspects.

- **Agent:** A computer program that can accept tasks and goals, can figure out which actions to perform in order to perform these tasks and can actually perform these actions without supervision. It is capable of performing a set of tasks and providing a set of services.
- **Role:** A collection of tasks that may be performed by a single entity.
- **Organization:** A group of agents working together to achieve a common purpose. An organization consists of roles that characterize the agents, which are members of the organization.
- **Society:** A collection of agents and organizations that collaborate to promote their individual goals.
- **Rule:** A guideline that characterizes a society. An agent that wishes to be a member of the society is required to follow all of the rules within.
- **Platform:** A physical node on which an agent can execute.
- **User:** A physical and environmental entity (usually a human) that interacts with an agent.

These building blocks can be used for constructing and improving agent-oriented modeling methods and serve as a benchmark for comparison [22] between such methods. Such a comparison can check whether, and to what extent, an existing modeling method addresses concepts expressed here as building blocks: Are all the concepts addressed? Are there any redundancies (i.e., do two elements in the method represent the same building block)? Are there any conflicts (i.e., does one element of the method represent one building block in a specific context and another building block in another context)?

We propose this set of building blocks as a basis for specifying and implementing MAS applications. Reviewing agent frameworks, methodologies, and architectures, we found that this set addresses the development needs of MAS. Many (if not all of the) building blocks can be modeled as an object (or a collection of such), however, the objective of defining these building blocks is to make a clear statement of what the abstract building blocks for developing MAS application are. These are needed in order to simplify the MAS designer's work. We do not claim that these building blocks are the ultimate abstraction of MAS elements, although it is based on widely used elements across several frameworks and methodologies. Each of the existing modeling methods for MAS indeed provides concepts for modeling MAS, however, they do not provide definitions of building blocks for MAS. Hence, we could not compare the proposed set to these alternatives. Yet, a set of building blocks for intelligent agents was introduced by [23]. These have some similarities to our proposed building blocks. However, that study differs from ours in the following: (1) the building blocks presented in that research are related to single agent applications while in our work we relate to MAS as well; (2) that research refers to BDI-based agents while we are trying to tackle various types of agent systems; and (3) that research does not show how these building blocks are integrated in a model of an agent application as we demonstrate in the proceeding sections.

3. OPM AND OPM/MAS

Object-Process Methodology (OPM) [4] is an integrated approach to the study and development of systems in general and software systems in particular. The basic premise of OPM is that objects and processes are two types of equally important classes of things, which together describe the function, structure and behavior of systems in a single framework in virtually any domain. OPM unifies the system lifecycle stages—specification, design and implementation—within one frame of reference, using a single diagramming tool – a set of Object-Process Diagrams (OPDs) and a corresponding subset of English, called Object-Process Language (OPL).

In OPM, an object class can be either systemic (internal) or environmental (external to the system). Orthogonally, it can be either physical or informatical (logical). An object class can be at one of several states, which are possible internal status values of the class objects. At any point in time, each object is at some state, and objects are transformed (generated, consumed, or change their state) through the occurrence of a process. A process can affect (change the state of) an environmental or a physical device, or the state or value of a specific attribute of an object class (rather than the state of the entire class, as in object-oriented methods).

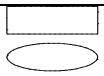
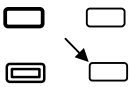
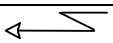
Unlike the object-oriented approach, behavior in OPM is not necessarily encapsulated within a particular object class construct. Using stand-alone processes, one can model behavior that involves several object classes and is integrated into the system's structure. Processes can be connected to the involved object classes through procedural links, which are divided, according to their functionality, into three groups: enabling links, transformation links, and control links. OPM also supports the definition of multiple scenarios which use the same entities. This is done by labeling paths over the procedural links. OPM provides this capability in order to remove the ambiguity arising from multiple outgoing procedural links.

OPM's built-in scaling (refinement/abstraction) mechanisms, which are unfolding/folding and in-zooming/out-zooming, help manage system complexity. Unfolding/folding is applied by default to objects for detailing/hiding their structural components (parts, specializations, features, or instances). In-zooming/out-zooming is applied by default to processes for detailing/hiding their sub-process components and details of the process execution. A third scaling mechanism, state expression/suppression, provides for showing or hiding the states of an object class. OPM's scaling mechanisms facilitate focusing on a particular subset of things (objects and/or processes), elaborating on the details of each one by refining each thing to any desired level of detail, and managing the complexity of a system model.

The ability to use a single OPM model for specifying the important system aspects—function, structure and behavior—prevents compatibility and integration problems among different diagram types by avoiding the model multiplicity problem [17]. Object-Process Language (OPL) is the textual counterpart of the graphic representation through OPDs. It is a constrained natural language that is intelligible to executives and domain-experts who are usually not familiar with system modeling methods, while at the same time amenable to machine compilation for code generation. In summary, the OPM framework expressed visually by the OPD-set and textually by the OPL script, provides the ability to grasp the entire system through one visual language,

which is useful primarily for system architects and developers, and one textual language. Table 1 shows the main OPM concepts, including its OPM/T [16] and OPM/Web [18] enhancements, which are used in this paper.

Table 1. OPM Concepts

Concept Name	Symbol	Definition
Object		An entity that has the potential of stable, unconditional physical or mental existence.
Process		A pattern of transformation that objects undergo.
Essence		An attribute that determines whether the thing (object or process) is physical or informational.
Affiliation		An attribute that determines whether the thing is environmental or internal.
Initial/ Regular/ Final/ Default State		A situation at which an object can exist for a period of time.
Exhibition-Characterization		A fundamental structural relation from a thing (object or process) or a tagged structural relation to a feature (attribute or operation) it exhibits.
Generalization-Specialization		A fundamental structural relation, which denotes the fact that a thing generalizes one or more specialized things.
Aggregation-Participation		A fundamental structural relation, which denotes the fact that a thing aggregates (i.e., consists of, or comprises) one or more things, each of which is referred to as a part of it.
Tagged structural relationship		An association between two things that holds true in the system irrespective of the time of inspection.
Instrument link		A procedural link indicating that a process requires an object for its execution (has a "wait until" meaning).
Effect link		A procedural link indicating that a process changes an object.
Result/ Consumption link		A procedural link indicating that a process creates/consumes an object.
Invocation link		A procedural link indicating that a process activates (invokes) another process.
Event link		A procedural link representing an event (data change, external, etc.) which activates a process.
Condition link		A procedural link representing a condition required for a process execution (has an "if" meaning).
Agent link		A procedural link indicating that a human actor is required for a process execution.

3.1 The MAS extension for OPM

AOSE methods suffer form lack of support in software engineering aspects such as accessibility and expressiveness. To overcome these limitations, we base our method for specifying MAS application on OPM, as its expressiveness and accessibility have been demonstrated [17]. However, since OPM is a general-purpose methodology, using its core symbol set may be too low-level for modeling a domain-specific application. This is so because the pertinent domain uses specialized concepts and building blocks that are at a higher level than the basic OPM entities (objects, processes, and states).

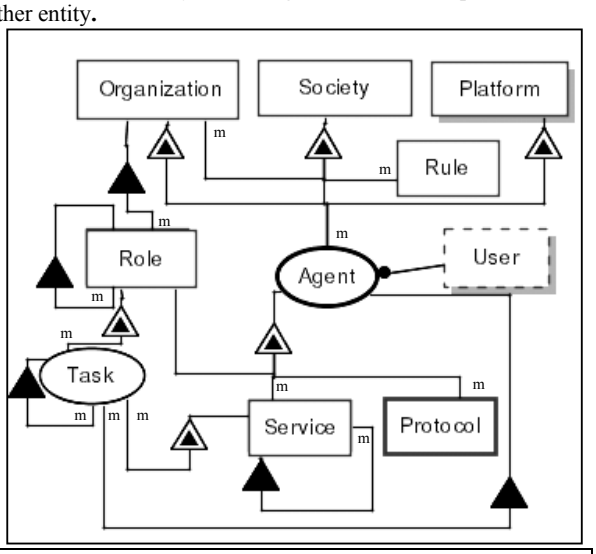
OPM/MAS follows principles of the Meta Object Facility (MOF) concept [13], which enables its flexibility. In this work we define MOF equivalent four layers as follows: (1) the meta-metamodel, which is the OPM itself; (2) the metamodel, which describes the specific building blocks within the MAS domain using OPM; (3) the model, which is based on the metamodel; and (4) the application, which is an implementation of the model.

OPM, as the first layer, was introduced in section 3. In this section we introduce the second layer which is the metamodel of the MAS domain. This metamodel was built based on common understandings of the agents research community, such as in [5], [10], and [14], and uses the MAS domain building blocks defined in Section 2.

In Figure 1, the top level of the metamodel for MAS is described. The description includes two presentations: the first one is the graphical representation via an OPD and the second one is the equivalent textual representation using OPL. This metamodel explains our approach towards MAS application development and defines the relationships between the different building blocks. For example, an **Agent** consists of **Tasks** and exhibits **Services**, **Protocols** and **Roles**. A **Society** exhibits **Organizations**, **Agents** and **Rules**. We do not provide further explanations of Figure 1 as these are provided by the associated OPL script, and are similar to the explanations provided regarding the **Agent** process and the **Society** object.

We divide the set of MAS building blocks into two groups. The first group contains static, declarative building blocks, while the second group contains building blocks with behavioral, dynamic nature. The building blocks in the first group, which include organization, society, platform, rule, role, user, protocol, belief, desire, fact, goal, intention, and service, are modeled as OPM *objects*. For example, role is a building block that defines a set of tasks that it uses. This is explicitly defined in the OPD of Figure 1. The building blocks in the second group, which include agent, task, and messaging, are modeled using the *process* concept. For

example, agent is an autonomous entity that may characterize an organization, a society or a platform, but is not part of them (see Figure 1). Agent has its attributes of role, service, protocol, objects and facts (see Figure 1 and Figure 2). Further, the agent consists of tasks and messaging processes. These characteristics exhibit the autonomy of the agent, as it is independent of any other entity.



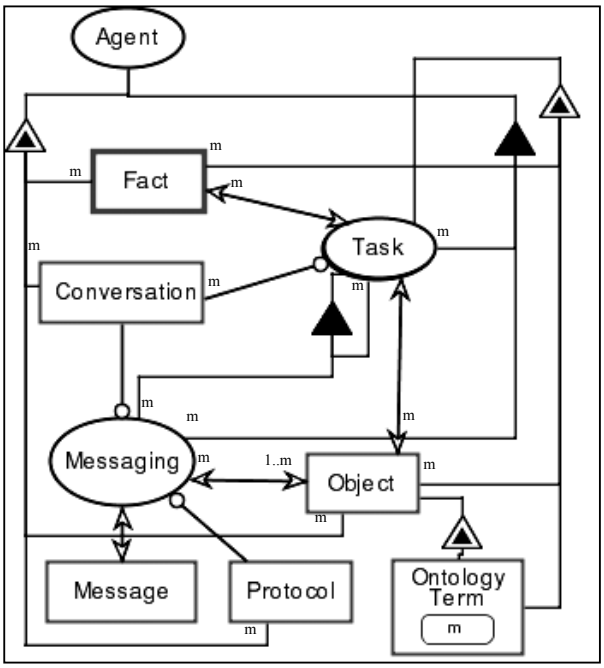
A **User,** which is physical and environmental, handles an **Agent** Process.
A **Platform,** which is physical, exhibits many **Agent** Processes.
A **Society** exhibits many **Rules,** many **Organizations,** and many **Agent** Processes.
An **Organization** exhibits many **Agent** Processes.
An **Agent** Process exhibits many **Roles**, many **Protocols** and many **Services**.
An **Agent** Process consists of many **Task** Processes.
An **Organization** consists of many **Roles.**
A **Role** exhibits many **Task** Processes.
A **Task** Process consists of many **Task** Processes.
A **Role** consists of many **Roles.**
A **Service** consists of many **Services**.
A **Service** exhibits many **Tasks**.

Figure 1. Top Level OPD of the Metamodel for MAS

In the following, we shortly discuss the advantages of the OPM/MAS metamodel over the UML metamodel. The OPM/MAS metamodel, which is based on OPM ontology, expresses both the structure and the behavior of the MAS domain in a single model. A UML-based metamodel, in contrast, handles only the structural relationships between the elements. For example, the OPM/MAS metamodel defines the effects that executing a task may have on objects and facts. Although UML uses many diagram types, specifying something like this in a UML metamodel may be very tedious if at all possible. In UML, the class diagram depicts the system's structure and knowledge organization, the sequence diagram depicts a single scenario, the Statecharts depict the reactive nature of the system, and the deployment diagram depicts its physical architecture. OPM unifies these aspects in a single model, and this increases the model expressiveness (i.e., the important system aspects are integrated in a single model) and accessibility (i.e., there is just one model and far less symbols to learn and use).

In the OPD in Figure 2 the agent in unfolded and, as before, this is specified both graphically and textually. For example, an agent consists of **Tasks** and **Messaging** processes and exhibits **Objects**, **Facts**, and **Protocols**. Note that the agent characteristics and parts are not identical in the OPDs of Figure 1 and Figure 2. To keep each OPD simple, OPM enables selective refinement of things. For example, in Figure 1 the **Agent** process does not consist of a **Messaging** process, while in Figure 2 it does. The complete specification is the union of details in the entire OPD set.

The multiplicity (m) symbols that appear on the links indicate the relationship cardinality between the building blocks. A multiplicity on either side of the link can be one of the following: zero or one, exactly one, at least one, and many. For instance, a task may interact with many facts (or none) and a fact should interact with at least one task.



An **Agent** Process exhibits many **Conversations**, many **Objects,** many **Protocols,** and many **Facts**.
An **Object** exhibits **Ontology Term**.
An **Ontology Term** can be one of many values.
An **Agent** Process consists of many **Task** Processes and many **Messaging** Processes.
Task Process exhibits **Ontology Term**, many **Objects**, and many **Facts**.
Task Process consists of many **Task** Processes and many **Messaging** Processes.
Task Process requires many **Conversations**.
Task Process affects many **Objects** and many **Facts**.
Messaging requires a **Conversation** and a **Protocol**.
Messaging affects a **Message** and at least one **Object**.

Figure 2. The unfolded OPD of Agent

Due to a lack of space, we provide only part of the OPM/MAS metamodel. However, it is worth noting that a **Fact** may be a **Belief**, a **Desire**, or an **Intention** and a **Desire** may be a **Goal**.

To this end we only discussed the OPM-based metamodel of our method. The next layer is the model. The model follows the rules determined within the metamodel. Thus, the metamodel serves two related purposes. One is to define domain-specific building blocks or components that are comprised lower-level entities

(objects, processes, and states). This will enhance the reuse of the MAS building blocks and prevent the need to reinvent them from scratch. The second purpose of the metamodel is to verify the correct use of the building blocks and their relationships within the model.
The linkage between the metamodel and the model is done via the domain labels. These labels are recorded in the upper left side of an entity of each thing within the model.
In this paper, we will not refer to the fourth layer of MOF—the application layer—because it refers to the implementation issues, which are not discussed in this paper.

4. THE FRUIT MARKET EXAMPLE

In this section we present the Fruit Market example and model part of the application using our MAS metamodel. The example demonstrates the core concepts of OPM/MAS and the use of its single-, unified model, but it definitely does not cover all of OPM/MAS capabilities.

4.1 The Fruit Market Specification

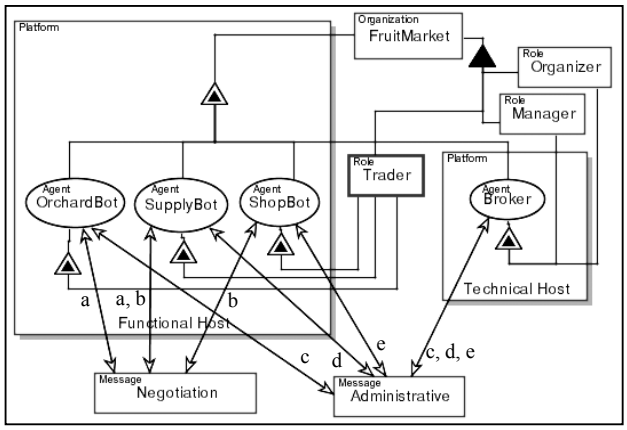
The Fruit Market example [2] describes a sample marketplace application whose objective is to enable the market participants to electronically trade fruit over a network via agents as part of a supply chain process. The following four participant types are agents in the Fruit Market.

- **OrchardBot** is an agent representing an orchard owner.
- **SupplyBot** is an agent representing a fruit producer
- **ShopBot** is an agent representing a supermarket chain.
- **Broker** is an agent representing a human broker who manages the information about each participant and shares this information with the participants.

We assume that the market is open, so agents can join or leave at any time, and agents are initially unaware of their counterparts.

4.2 The Fruit Market Model

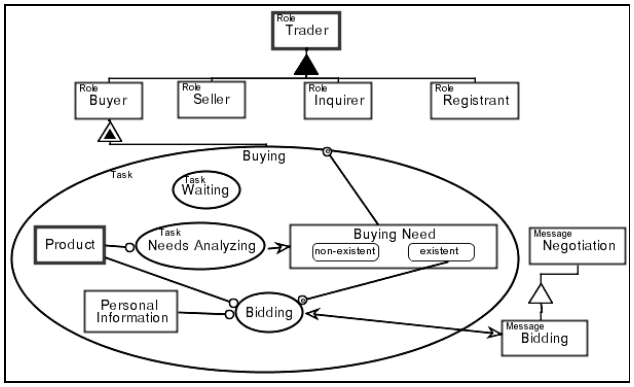
Based on the MAS metamodel, the model of the Fruit Market system is presented in Figure 3, which is the system diagram (top-level OPD), and Figure 4, which is an unfolded OPD of an agent. As discussed before, the linkage between the metamodel and the application model is done via the domain label.
In this case study we demonstrate the dual use of the metamodel: the building blocks reuse and the rules validation. For example, the **FruitMarket** organization consists of the **Trader**, **Manager** and **Organizer** roles. The **FruitMarket,** labeled as organization, exhibits the agents **OrchardBot**, **SupplyBot**, **ShopBot** and **Broker**. These entities follow the rules depicted in Figure 1, such that a role is a part of an organization. In addition, the two hosts of the system: **Functional Host** and **Technical Host** are described along with their associated agents. Path labels on the links that are connected to the **Negotiation** Message and **Administrative** Message indicate the interaction between the agents. For example, by following the paths labeled '**a**' and '**c**', the **OrchardBot** agent interacts with the **SupplyBot** agent and the **Broker** agent, respectively.
In the OPD presented in Figure 4, the **Trader** Role is unfolded. It consists of four roles. The **Buyer** Role exhibits (among other tasks) the **Buying** Task. The **Buying** Task zooms into (consists of) three lower-level tasks the executed order of which is determined, as in any OPD, by their position along the vertical axis, along which time flows from top to bottom. For example, the **Needs Analyzing** Task requires a **Product** and yields a **Buying Need**, while the **Bidding** Task requires a **Product** and **Personal Information** and affects a **Bidding** Message, which is a **Negotiation** Message. In addition, the **Bidding** Task is triggered when the **Buying Need** enters the state of **existent.**



A **FruitMarket** Organization consists of a **Trader** Role, a **Manager** Role and an **Organizer** Role.
A **Functional Host** Platform exhibits an **OrchardBot** Agent, a **SupplyBot** Agent and a **ShopBot** Agent.
An **OrchardBot** Agent exhibits a **Trader** Role.
Following path **a**, an **OrchardBot** Agent affects **Negotiation** Message**.**
Following path **c**, an **OrchardBot** Agent affects **Administrative** Message**.**
A **SupplyBot** Agent exhibits a **Trader** Role.
Following path **a**, a **SupplyBot** Agent affects **Negotiation** Message.
Following path **b**, a **SupplyBot** Agent affects **Negotiation** Message.
Following path **d**, a **SupplyBot** Agent affects **Administrative** Message.
A **ShopBot** Agent exhibits a **Trader** Role.
Following path **b**, a **ShopBot** Agent affects **Negotiation** Message.
Following path **e**, a **ShopBot** Agent affects **Administrative** Message.
A **Technical Host** Platform consists of a **Broker** Agent.
A **Broker** Agent exhibits a **Manager** Role and an **Organizer** Role.
Following path **c**, a **Broker** Agent affects **Administrative** Message.
Following path **d**, a **Broker** Agent affects **Administrative** Message.
Following path **e**, a **Broker** Agent affects **Administrative** Message.

Figure 3. Fruit Market – Top Level OPD

This example demonstrates the system architecture, which comprises two hosts, the agent types within the system, the role hierarchy, and one of the role's tasks. The tasks specify the agents' behavior. This system is specified solely by the single bimodal, graphic-textual OPM/MAS model. Modeling the same case study in UML would require at least four diagrams: deployment diagram to specify the hosts and their agents, class diagram to describe the agents' structure, attributes and operation, a sequence diagram to show the behavior of each task (hence several sequence diagrams may be required), and a Statecharts diagram to specify (for example) the event the occurs when **Buying Need** enters **non-existent** (i.e., when it triggers the buying process).



A **Bidding** Message is a **Negotiation** Message.
A **Trader** Role consists of a **Buyer** Role, a **Seller** Role an **Inquirer** Role and a **Registrant** Role.
A **Buyer** Role exhibits a **Buying** Task.
Buying Task exhibits **Product**, **Personal Information** and **Buying Need**.
Buying Need can be **existent** or **non-existent**.
Buying Task consists of a **Waiting** Task, a **Needs Analyzing** Task and a **Bidding** Task.
Needs Analyzing Task requires a **Product**.
Needs Analyzing yields a **Buying Need**.
Bidding Task requires **Product** and **Personal Information**.
Bidding Task affects **Bidding** Message.
Buying Need triggers **Buying** Task when it enters **non-existent**.
Buying Need triggers **Bidding** Task when it enters **existent**.

Figure 4. Unfolding of Trader Role

5. EXISTING AGENT INFRASTRUCTURE

Designers of MAS may wish to base their MAS application on a specific exiting infrastructure. Such infrastructure consists of ready-made components that the designer would like to weave into the application specification (model). A MAS modeling method should be able to support this capability. Using the RETSINA architecture[15] as an example for MAS infrastructure, in this section we discuss the way in which existing components are weaved into an application model based on OPM/MAS.

We divide the RETSINA components into two groups: application data and generic components[4]. The application data group consists of the following components:

- The objective DB is a dynamic store that holds the objectives of the agent. The objectives can be modeled in OPM/MAS as **goals**.
- The task schema library and task reduction library are data stores that hold the task schemas and "sub-plans". These can be modeled in OPM/MAS using the **task** building block and its hierarchy to depict the hierarchical task network within RETSINA.
- The beliefs DB is a dynamic data store that maintains the agent's knowledge of the domain in which the agent's plan will be executed. The beliefs can be modeled in OPM/MAS as **beliefs**.

The generic components are the planner, the scheduler, the execution monitor, and the communicator. These components can be modeled using the core OPM, presented in Section 3. When modeling a specific application, one can reuse the models of the generic components and weave the specific application data model into it. Figure 5 shows the system diagram of a RETSINA-based system. It consists of the four generic OPM processes, which are correspond to the four generic components and their relationships with the application data group.

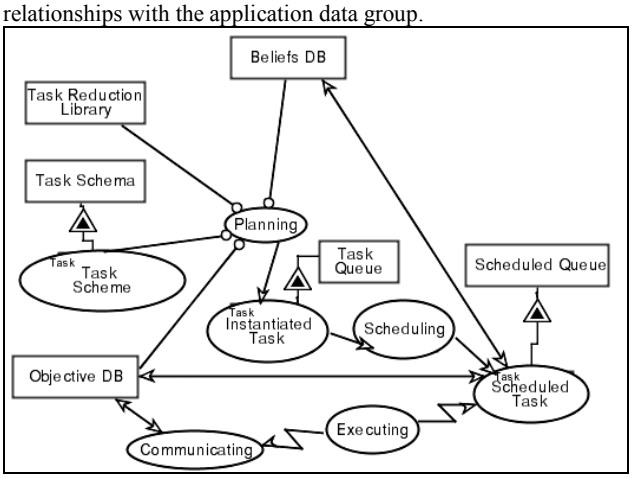


Figure 5. The system diagram of the RETSINA architecture

The **Planning** process moves the required **Task Schemes** into the **Task Queue** according to the **Belief DB** and the **Objective DB**. The **Scheduling** process moves the required **Instantiated Tasks** into the **Scheduled Queue**. The **Executing** process activates the **Scheduled Task** and the **Communicating** process when required. The **Communicating** process affects the **Objective DB**. The **Scheduled Task** process affects the **Objective DB** and the **Belief DB**.

As this example shows, OPM and its MAS extension can be used to specify both the ready-made artifacts (the generic components) and the new application elements (the application data), into single model.

6. CONCLUSION

In this paper, we have addressed two problems within the AOSE domain: the lack an agreed upon core MAS components and the lack of sufficient expressiveness and accessibility within the existing modeling methods for MAS.

To solve the first problem, we examined existing MAS modeling methods and based on our insights we define a set of building blocks for modeling MAS applications. This set can serve as a basis for developing agent-oriented modeling methods, as well as a benchmark for comparison between various MAS modeling methods and frameworks. In response to the second problem of lack of expressiveness and accessibility, we introduce a new approach to MAS modeling based on the Object-Process Methodology (OPM). The benefits of OPM/MAS are the following: increased accessibility by an intuitive bimodal (textual and graphical) model; increased expressiveness due to the unification of the structure and the behavioral aspects of a system; increased flexibility by enabling to easily change the metamodel; and increased support in software engineering criteria, such as documentation and code generation. Applying this approach to the domain of multi-agent systems (MAS), we define an OPM/MAS metamodel and use it to model a sample agent application. We also demonstrated how to model an application which is based on an existing infrastructure using OPM and OPM/MAS.

[4] We refer to components as generic when they can be (re-)used across multiple applications without modifications.

The accessibility and expressiveness of OPM were evaluated and found to be better than in an object-oriented based method. Nevertheless, since OPM/MAS is in its initial stage of development, more testing and evaluation are required, and are currently performed. This should be done in order to further prove the suitability of OPM/MAS to the construction of MAS.
OPM/MAS supports the lifecycle development stages of capturing the requirements, the analysis and the design of MAS. In order to support the stages of implementation and testing as well, we are developing an infrastructure that is based on the OPM/MAS metamodel. This infrastructure will automatically generate application code far beyond a code skeleton.

7. REFERENCES

[1] F. M. T. Brazier, B. Dunin-Keplicz, N. R. Jennings and J. Treur, DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. Intl. Journal of Cooperative Information Systems, Vol. 6, pp. 67-94, 1997.

[2] British Telecom, Zeus, http://www.btexact.com/projects/agents/zeus/

[3] S. A. DeLoach, M. F. Wood and Cl. H. Sparkman, Multiagent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, Vol. 11 (3), pp. 231-258, 2001.

[4] D. Dori, Object-Process Methodology - A Holistic Systems Paradigm, Springer, Heidelberg, 2002.

[5] M. Fisher, J. Muller, M. Schroeder, G. Staniford, and G. Wagner, Methodological foundations for agent-based systems, Knowledge Engineering Review (12) 3, pp. 323-329, 1997.

[6] S. J. Franklin, Assessing the suitability of UML for Capturing and Communicating Systems Engineering Design Models, Vitech Coroporation, www.vtcorp.com/infocenter/ SuitabilityOfUMLForSE_2002.pdf, 2002.

[7] C. A. Iglesias, M. Garrijo, J. Gonzalez and J. R. Velasco, Analysis and Design of multiagent systems using MAS-CommonKADS, Intelligent Agents IV: Agent Theories, Architectures and Languages, M. P. Singh, Anand Rao and M. J. Wooldridge, eds., LNCS 1365, Springer, pp. 313-328, 1997.

[8] C. A. Iglesias, M. Garijo, and J. C. Gonzalez. A survey of agent-oriented methodologies. In J. P. Muller, M. P.Singh, and A. S. Roa, (eds.), Intelligent Agent V, Proc. of ATAL-98, LNCS 1555, Springer, pp. 317-330, 1999.

[9] N. R. Jennings, K. P. Sycara, and M. Wooldridge A Roadmap of Agent Research and Development In Journal of Autonomous Agents and Multi-Agent Systems, Vol.1(1), pp. 7-36. July 1998.

[10] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, B. Odgers and J. L. Alty, Implementing a Business Process Management System using ADEPT: A Real-World Case Study, Intl. Journal of Applied AI 14 (5), pp. 421-465, 2000.

[11] M. Luck and M. d'Inverno, Structuring a Z Specification to Provide a Formal Framework for Autonomous Agent Systems, Proc. of ZUM'95, J. Bowen and M. Hinchey (eds.), LNCS 967, Springer, pp. 47-62, 1995.

[12] P. Massonet, Y. Deville and C. Neve, From AOSE Methodology to Agent Implementation, Proc. 1st Joint Conference on Autonomous Agents and Multi-Agents Systems, pp. 27 – 34, 2002,

[13] OMG, Meta Object Facility (MOF) v1.3.1, http://www.omg.org/technology/documents/formal/mof.htm

[14] J. Odell and C. Bock, Suggested UML Extensions for Agents, Response to the OMG Analysis and Design Task Force UML RTF 2.0 Request for Information, 1999.

[15] M. Paolucci, O. Shehory, K. Sycara., D. Kalp, and A. Pannu,. A Planning Component for RETSINA Agents. LNAI 1747, pp. 147-161, 1999.

[16] M. Peleg and D. Dori, Extending the Object-Process Methodology to Handle Real-Time Systems, Journal of Object-Oriented Programming, 11(8), pp. 53-58, 1999.

[17] M. Peleg and D. Dori, The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods IEEE Transaction on Software Engineering, 26(8), pp. 742-759, 2000.

[18] Reinhartz-Berger, S. Katz, D. Dori, OPM/Web - Object-Process Methodology for Developing Web Applications, Annals on Software Engineering - Special Issue on OO Web-based Software Engineering, pp. 141-161, 2002.

[19] A. S. Rao and M. P. Georgeff, Modeling rational agents within a BDI-architecture. Proceedings of Knowledge Representation and Reasoning, pp. 473-484, 1991

[20] K. Siau and Q. Cao., Unified Modeling Language: A Complexity Analysis, Journal of Database Management, pp. 26-34, 2001.

[21] O. Shehory and A. Sturm, Evaluation of Modeling Techniques for Agent-Based Systems, Proceeding of The Fifth International Conference on Autonomous Agents, pp. 624-631, 2001

[22] Y. Wand and R. Weber, An ontological Model of an Information System, IEEE Transaction on Software Engineering, Vol. 16, No. 11, pp. 1282-1292, 1990.

[23] M. Winikoff, L. Padgham, and J. Harland, Simplifying the Development of Intelligent Agents, Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI'01), pages 557-568, 2001.

[24] M. Wooldridge, N. R. Jennings, and D. Kinny, The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and MAS, Vol. 3 (3), pp. 285-312, 2000.