

An Object-Process-Based Modeling Language for Multi-Agent Systems

ARNON STURM*

Department of Information Systems Engineering, Ben Gurion University of the Negev, Beer Sheva, 84105, Israel
sturm@bgu.ac.il

DOV DORI

Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, 32000, Israel, and Engineering Systems Division, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
dori@mit.edu

ONN SHEHORY

IBM Haifa Research Lab, Haifa University Campus, Haifa, 31905, Israel
onn@il.ibm.com

Abstract

While a number of modeling languages for constructing Multi-Agent Systems (MAS) have been suggested, none of them is widely accepted and used. A prominent reason for this is the gap that exists between agent-oriented modeling languages and the agent-based system modeling needs, including accessibility, flexibility, and expressiveness. This paper addresses the need for such a language by proposing OPM/MAS, an agent modeling language that extends Object-Process Methodology (OPM) with an intermediate metamodel of the MAS domain. Three case studies and a comparison to contemporary agent modeling languages demonstrate the novelty and benefits of OPM/MAS.

Keywords: agent-oriented software engineering, modeling language, agent-based systems, multi-agent systems

1. Introduction

Over the last decade, researchers and practitioners have recognized the advantages of applying the agent paradigm to system development. Yet, the number of deployed commercial agent-based applications is rather small. One of the reasons for this slow technology transfer is the lack of agreement on what a modeling language for agent-based applications should look like [33, 36, 68]. The software engineering (SE) research community widely agrees that a suitable modeling language is an essential element of systems development. The language provides the basis for specifying the desired system. If this language is not too technical, it can facilitate communication between system developers and business customers, and provide a basis for high-quality documentation. Recognizing these benefits, many MAS¹ development methodologies (including methods and modeling languages) that follow SE principles have been developed, including ADELFE [4], ADEPT [35], AUML [2], CAMLE [50], DESIRE [5], FAF [18], Gaia [67], INGENIAS [30, 45, 46], MAS-CommonKADS [34], MaSE [15, 16, 17, 51], MESSAGE [7], Prometheus [65], and TROPOS [6, 8].

Despite the proliferation of MAS modeling languages, none of them has been widely adopted by the potential user community. Studies that examined and compared the properties of existing modeling languages [3, 32, 52, 58] have suggested that none of these languages is fully suitable for the development of MAS. Although these methods are based on solid logical and agent-oriented foundations, their support for important software engineering characteristics of modeling languages, notably accessibility, expressiveness, and flexibility, is not sufficient [52]. Additionally, models generated by existing methods do not fulfill the requirements of platforms available for MAS implementation [63]. These deficiencies have had an adverse effect on industry acceptance and adoption of agent technology [66].

To gain better understanding of these deficiencies, we hereby revisit common definitions of the accessibility, expressiveness, and flexibility properties in the context of SE methodologies. The accessibility property, also known as ease of encoding, refers to the ease, or simplicity, of understanding and using a modeling language [54]. The expressiveness property, also known as expressive adequacy, refers to the capability of the modeling language to present system concepts at various abstraction levels and with various system aspects, notably structure, behavior, and interaction [54]. The flexibility property refers to the ability to change or adjust the modeling language according to the organization or designer's needs. Since there is practically little agreement across methods and languages on the concepts formalizing MAS, developers using them may need to specify various sets of these concepts. Hence, method

¹ The use the acronym MAS to refers to both multi-agent systems and single agent systems as a special case.

and language flexibility are needed. Clearly, limited support of these three properties has an adverse effect on the usability and acceptance of a software engineering method and a modeling language associated with it.

Studying the domain of MAS, we are specifically interested in properties of agent-oriented modeling languages. In this domain, accessibility of existing modeling languages is limited, as these languages are complex, they consist of multiple views, and lack complexity mechanisms for managing the models. The expressiveness of these modeling languages is sometimes limited too, since they focus on fulfilling agent-based systems functions, such as knowledge management, often at the expense of appropriate catering to other SE needs. In other cases, agent modeling languages lack sufficient expressiveness to specify implementation details. The flexibility of contemporary agent modeling languages is also limited. In particular, many of them have rigid metamodels, which cannot be changed by potential users. As a consequence, users often define their own methodologies and modeling languages instead of reusing existing ones [12].

To address the limited support of some SE properties by MAS modeling languages, one may opt for one of three major approaches: (1) develop a new MAS modeling language that overcomes the limitations of existing ones, (2) unify existing methods, or (3) adapt an existing systems modeling language for MAS. Developing a new MAS modeling language from scratch requires extensive validation efforts. Unifying existing methods allows agent-based system designers to construct their own methodology [3, 11, 12]. Method unification has some weaknesses and modeling languages are difficult to integrate and glue [44]. In view of this, we have followed the path of adapting an existing general-purpose modeling language to the specific needs of modeling MAS. By utilizing an existing language, one can capitalize on proven advantages of the adopted language and significantly reduce the effort required for developing a new language. However, one should bear in mind that this might introduce some new problems into the modified language.

Next, we explain the selection of the modeling language that serves as the basis upon which our new MAS modeling language is constructed. Current general-purpose MAS modeling languages can be classified by their modeling approach into two classes: knowledge engineering oriented languages and software engineering oriented languages [33]. Knowledge engineering oriented languages emphasize the specification of knowledge acquisition and reasoning, whereas software engineering oriented languages focus on the specification of the system structure and behavior, taking into consideration aspects such as reusability and maintenance. Knowledge engineering based agent-oriented modeling languages suffer from weaknesses in modeling and in supporting reuse. These languages also exhibit limited accessibility and expressiveness [52]. These weaknesses suggest that one should not select a knowledge engineering

based agent-oriented modeling language as the basis for developing a general purpose MAS modeling language, since with this type of language as a starting point, addressing the particular software engineering aspects of interest might be too difficult. In contrast, software engineering agent-oriented modeling languages do provide agent-oriented abstractions using object-oriented concepts, which support the notions of modeling and reuse.

Initial attempts at building agent-oriented languages based on object-oriented concepts have indeed considered an existing object-oriented language – the Unified Modeling Language (UML) [43]. Apparently, UML is a leading candidate for becoming a basis for a MAS development modeling language. While UML has been adopted as a basis for AUML [2], it combines a rather large number of diagram types [53], introducing a high level of complexity. As a result, using it as a basis for a MAS modeling language would leave some of the above stated problems unresolved. Moreover, as discussed in [29], the consistency and integrity of UML are questionable. Whereas AUML enhances the expressiveness of UML for specifying MAS, it does not address the accessibility aspect, namely the ease of constructing models and understanding existing ones². There are other suggestions for multi-agent system specification languages, such as Agent Modeling Language (AML) by Whitestein Technologies [9] and CAMLE [50]. However, due to similar reasons, these languages exhibit limitations that are not unlike those introduced by AUML. For example, while CAMLE, which consists of multiple views, provides for consistency checking, it suffers from problems similar to those of UML and its variants.

With these limitations in mind, we have examined Object-Process Methodology (OPM) [20] as a basis for an alternative MAS modeling language. OPM is an integrated approach to the study and development of software systems and systems in general. 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. A features-based evaluation of OPM [56] has shown that OPM is well-suited for developing MAS applications. At least with respect to accessibility and expressiveness, which are the two major aspects the proposed modeling language aims to address, using OPM has gained better results than using the object-oriented paradigm [47, 49]. Another important feature of OPM is that it treats processes and objects on equal footing, eliminating the object-oriented supremacy of objects over processes. This is an essential requirement in the area of agent-based systems, as agents are behavioral entities. We do not claim that OPM is superior to UML. Rather, we

² One may claim the UML has become the standard de-facto modeling language. However, the main use of UML is attributed to the use of its class diagram [19].

present it as an alternative due to its novel notion of weaving the system structure and behavior. This combination has been proven experimentally to be helpful in constructing and understanding system specification [47].

Compared with object-oriented approaches, OPM is highly usable. However, due to its generic nature, expressing building blocks of a specific domain, such as MAS, may prove to be tedious. Additionally, such specifications may be inconsistent over time or across different developing communities. To overcome these limitations, this work extends OPM by adding an intermediate layer, in which semantics is clearly defined in terms of guidelines and constraints, allowing for MAS components to follow common modeling practices. This extension, called OPM/MAS, comprises a set of predefined building blocks that capitalize on the conciseness and expressiveness of OPM [59]. This paper presents OPM/MAS, demonstrates its application, and evaluates it.

The major advantage of OPM/MAS is its semi-formal yet intuitive single model approach. Expressed in graphic and equivalent natural language text, OPM/MAS overcomes accessibility and expressiveness limitations that characterize other MAS modeling languages, as we demonstrate in the sequel. The layered approach presented in this paper enables users to define their own building blocks. This is akin to changing the metamodel in a seamless manner, which increases the modeling language flexibility.

The rest of the paper is organized as follows. In Section 2, a set of MAS building blocks that enable the specification of agent-based system is introduced and defined. Section 3 presents OPM, and Section 4 describes the MAS extension of OPM. Section 5 demonstrates the use of OPM/MAS for specifying three multi-agent systems, representing various aspects of MAS. In Section 6, we evaluate the proposed modeling language, and Section 7 summarizes and concludes this work.

2. MAS BUILDING BLOCKS

Although the research field of Agent-Oriented Software Engineering (AOSE) has been active for well over a decade, there is no definition of or agreement on the set of standard elements within a language that are essential for constructing MAS. Early attempts to construct a unified metamodel for agent-oriented systems that relies upon building blocks [11] were made by groups within FIPA [26] via the Methodology Technical Committee and Modeling Technical Committee, and by AgentLink [1] via the Agent-Oriented Software Engineering Technical Forum Group. The objective in both cases were to establish a standard set of building blocks that would enhance coherence among agent-based systems and foster cross-fertilization among agent-based systems developers and researchers.

The set of MAS building blocks presented in this work is one of several possible building blocks sets. It reflects insights from agent systems, methodologies and architectures. This is a set of building blocks that are designed for the construction of general purpose multi-agent systems, but they are not intended to cover all the aspects of agent-based system. The MAS building blocks are listed below along with their definitions. In many cases, the definition relies on prior definitions accepted in the MAS domain.

- **Object:** A data container [41].
- **Agent:** A computer program that is capable of performing a set of tasks and providing a set of services. Specifically, an agent can accept tasks and goals, figure out which actions to execute in order to perform these tasks, and perform these actions without supervision [40].
- **Environment:** The (logical) world in which agents operate. [27].
- **Message:** A means of exchanging objects between agents.
- **Task:** A piece of work to be performed subject to constraints, which can be assigned to an agent or performed by it via a role (defined below) [40].
- **Role:** An abstract representation, encapsulating a set of tasks, of an agent's function or identification within a group of agents [40].
- **Protocol:** An ordered set of messages that together define the admissible patterns of a particular type of interaction between agents or roles.
- **Mobilizing:** A process for supporting mobility [24, 51], which includes the following attributes:
 - **Type:** The mobilizing method (migration, cloning, or invocation).
 - **Location:** The location to which the agent should migrate.
 - **Priority:** The importance level of the mobilization. This attribute is useful for prioritizing locations in case the agent can move to more than one location.
 - **Result:** The outcome of the mobilizing process.
- **Platform:** A physical computational node which can execute (run) an agent.
- **Organization:** A group of agents playing various roles and working together to achieve a common purpose [40].
- **Society:** A collection of individual agents and organizations that collaborate to promote their own goals [36].
- **User:** An intelligent being (usually a human) in the environment that interacts with one or more agents.

As noted, the list above is not a complete or exhaustive set of the MAS building blocks. Rather, it demonstrates the ability of the proposed modeling language to model MAS. However, our comprehensive review has indicated that this set of building blocks includes the elements which are needed for practical implementations. This finding is in line with

the recent conclusions of the Agent-Oriented Software Engineering Technical Forum Group (AOSE TFG) with respect to a unified MAS metamodel [12].

3. OBJECT-PROCESS METHODOLOGY

Object-Process Methodology (OPM) [20] 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) that is intelligible to domain-experts and non-technical executives who are usually not familiar with system modeling languages.










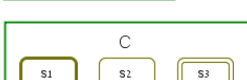
| ENTITIES | |  | |
|----------|---|--|---|
| Name | Symbol | OPL | Definition |
| Things | Object     Process   | B is physical. (shaded rectangle) C is physical and environmental. (shaded dashed rectangle) E is physical. (shaded ellipse) F is physical and environmental. (shaded dashed ellipse) | An object is a thing that exists. A process is a thing that transforms at least one object. Transformation is object generation or consumption, or effect—a change in the state of an object. |
| | State    | A is s1 . B can be s1 or s2 . C can be s1 , s2 , or s3 . s1 is initial. s3 is final. | A state is situation an object can be at or a value it can assume. States are always within an object. States can be initial or final. |

Figure 1 through Figure 4 specify the OPM graphical symbols—entities, structural relations, and procedural links, respectively. In OPM, an object or a process class can be either systemic (internal) or environmental (external to the system). In addition, it can be either physical or informatinal (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 an object is transformed (generated, consumed, or changes its 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 modeling languages).

Unlike the object-oriented approach, behavior in OPM is not necessarily encapsulated as an operation (or method) within a particular object class. Allowing the concept of a stand-alone process, one can model behavior that involves several object classes which are enabled and/or are transformed by the process class. Processes can be connected to the involved object classes through procedural links, which are classified into three groups: enabling links, transformation links, and control links. OPM also supports the definition of multiple scenarios which use the same entities (processes or stateful objects). This is done by labeling paths over procedural links, which remove the ambiguity arising from multiple outgoing/incoming procedural links.

OPM's built-in refinement/abstraction mechanisms, unfolding/folding and in-zooming/out-zooming, help in managing 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 exposing/hiding their sub-process components and details of the process execution. The time line inside an in-zoomed process is from top to bottom, so the execution order of the sub-processes within an in-zoomed process is determined by their top to bottom order of their graphical layout position. A third scaling mechanism, state expression/suppression, provides for showing or hiding the states of an object class. These OPM's scaling mechanisms facilitate managing the complexity of a system model by focusing on a particular subset of things (objects and/or processes), elaborating on the details of each entity by refining it to any desired level of detail while keeping track of the "big picture".


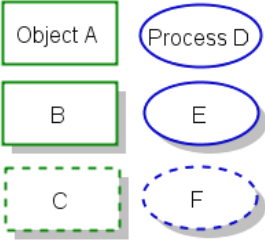
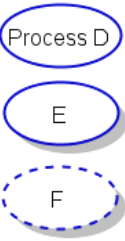
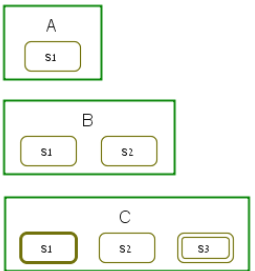

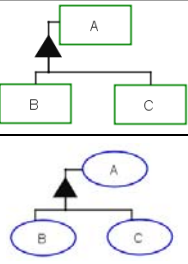
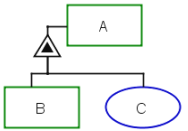
| ENTITIES | | |  | |
|----------|--|--|--|---|
| Name | Symbol | OPL | Definition | |
| Things | Object  | Process D  | B is physical. (shaded rectangle) C is physical and environmental. (shaded dashed rectangle) E is physical. (shaded ellipse) F is physical and environmental. (shaded dashed ellipse) | An object is a thing that exists. A process is a thing that transforms at least one object. Transformation is object generation or consumption, or effect—a change in the state of an object. |
| | State  | A is s1. B can be s1 or s2. C can be s1, s2, or s3. s1 is initial. s3 is final. | A state is situation an object can be at or a value it can assume. States are always within an object. States can be initial or final. | |

Figure 1. The OPM entities

| STRUCTURAL LINKS & COMPLEXITY MANAGEMENT | | |  |
|--|---|--|--|
| Name | Symbol | OPL | Semantics |
| Fundamental Structural Relations | Aggregation-Participation  | A consists of B and C. A consists of B and C. | A is the whole, B and C are parts. |
| | Exhibition-Characterization  | A exhibits B, as well as C. | Object B is an attribute of A and process C is its operation (method). A can be an object or a process. |

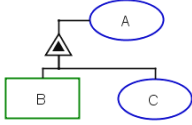
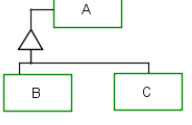
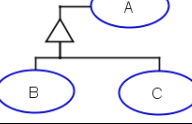
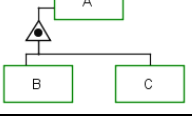
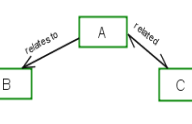
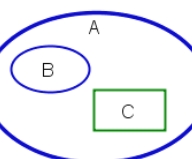
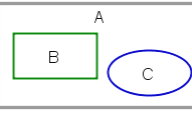
| | | | |
|--|---|---|--|
| |  | A exhibits B , as well as C . | |
| Generalization-Specialization |  | B is an A . C is an A . | A specializes into B and C. |
| |  | B is A . C is A . | A, B, and C can be either all objects or all processes. |
| Classification-Instantiation |  | B is an instance of A . C is an instance of A . | Object A is the class, for which B and C are instances. Applicable to processes too. |
| Unidirectional & bidirectional tagged structural links |  | A relates to B . (for unidirectional) A and C are related. (for bidirectional) | A user-defined textual tag describes any structural relation between two objects or between two processes. |
| In-zooming |  | A exhibits C . A consists of B . A zooms into B , as well as C . | Zooming into process A, B is its part and C is its attribute. |
| |  | A exhibits C . A consists of B . A zooms into B , as well as C . | Zooming into object A, B is its part and C is its operation. |

Figure 2. Structural links and Complexity Management

| ENABLING AND TRANSFORMING PROCEDURAL LINKS | | | | |
|--|----------------------------------|-----|-----------------------------------|---|
| Name | Symbol | OPL | Semantics | |
| Enabling links | Agent Link | | A handles B. | Denotes that the object is a human operator. |
| | Instrument Link | | B requires A. | "Wait until" semantics: Process B cannot happen if object A does not exist. |
| | State-Specified Instrument Link | | B requires s1 A. | "Wait until" semantics: Process B cannot happen if object A is not at state s1. |
| Transforming links | Consumption Link | | B consumes A. | Process B consumes Object A. |
| | State-Specified Consumption Link | | B consumes s1 A. | Process B consumes Object A when it is at State s1. |
| | Result Link | | B yields A. | Process B creates Object A. |
| | State-Specified Result Link | | B yields s1 A. | Process B creates Object A at State s1. |
| | Input-Output Link Pair | | B changes A from s1 to s2. | Process B changes the state of Object A from State s1 to State s2. |
| | Effect Link | | B affects A. | Process B changes the state of Object A; the details of the effect may be added at a lower level. |

Figure 3. Enabling and Transforming Procedural links


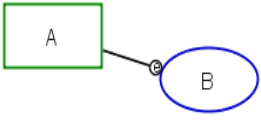
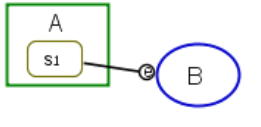
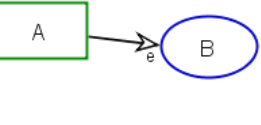
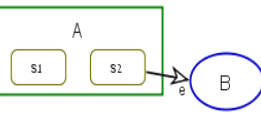
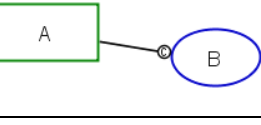
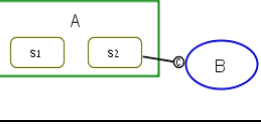
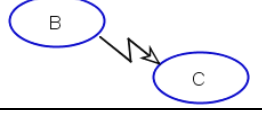
| EVENT, CONDITION, AND INVOCATION PROCEDURAL LINKS | | |  |
|---|---|---|--|
| Name | Symbol | OPL | Semantics |
| Instrument Event Link |  | A triggers B . B triggers A . | Existence or generation of object A will attempt to trigger process B once. Execution will proceed if the triggering failed. |
| State-Specified Instrument Event Link |  | A triggers B when it enters s1 . B requires s1 A . | Entering state s1 will attempt to trigger the process once. Execution will proceed if the triggering failed. |
| Consumption Event Link |  | A triggers B . B consumes A . | Existence or generation of object A will attempt to trigger process B once. If B is triggered, it will consume A. Execution will proceed if the triggering failed. |
| State-Specified Consumption Event Link |  | A triggers B when it enters s2 . B consumes s2 A . | Entering state s2 will attempt to trigger the process once. If B is triggered, it will consume A. Execution will proceed if the triggering failed. |
| Condition Link |  | B occurs if A exists. | Existence of object A is a condition to the execution of B. If object A does not exist, then process B is skipped and regular system flow continues. |
| State-Specified Condition Link |  | B occurs if A is s1 . | Existence of object A at state s2 is a condition to the execution of B. If object A does not exist, then process B is skipped and regular system flow continues. |
| Invocation Link |  | B invokes C . | Execution will proceed if the triggering failed (due to failure to fulfill one or more of the conditions in the precondition set). |

Figure 4. Event, Condition and Invocation Procedural links

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 [37, 47]. In summary, the OPM framework expressed visually by the OPD set and textually by the equivalent OPL paragraph set, provides the ability to grasp the entire system by its various stakeholders through the

visual language, which is useful primarily for system architects and developers, and its corresponding textual language, which is useful for domain experts. Following this idea, for the sake of clarity, in this paper, the OPM specification is presented using both presentations – OPD and OPL. A comprehensive specification of OPM is provided in [20, 48].

The OPD set is a scheme of the entire system or application, where each object and process represents a class rather than an instance. In case there is a need to refer to instances, OPM provides a special notation, shown in **Error!**
Reference source not found.

4. OPM/MAS: A MAS EXTENSION OF OPM

As discussed above, our extension of OPM has been developed to overcome limitations of current agent-oriented modeling languages, in particular the flexibility aspect. OPM was elected as the basis for OPM/MAS because of its recognized expressiveness and accessibility [47]. Since OPM is a general-purpose modeling language, using exclusively its low-level core symbol set may be too tedious for modeling a domain-specific application, such as MAS. This is so because any specific domain, with MAS being no exception, uses specialized concepts and building blocks that are at a higher level of abstraction than stateful objects and processes, the basic OPM entities.

Following the idea of metamodel layers, such as the Meta Object Facility (MOF) [42], OPM/MAS is defined over the following four layers: (1) M2: the meta-model, which is OPM in its generic vanilla form; (2) M1.5: the intermediate meta-model, which describes the specific building blocks within the MAS domain using OPM; (3) M1: the model, which follows the rules specified within the meta-models (from M1.5 and M2); and (4) M0: the application, which is an implementation of the M1 model. The architecture of OPM/MAS is somewhat different than the “classical” metamodel approach, as it does not use different levels of abstraction among M1, M1.5, and M2 layers. Specifically, in OPM/MAS, elements from M2 and M1.5 may appear in M1. This provides for keeping the modeling task similar across the intermediate metamodel layer and the model layer. We anticipate users of OPM/MAS to operate at both layers. M1.5 will be used to define the required agent notions, building blocks, notations, and the relationships among them, while M1 will serve to model the system using the elements defined in M1.5 (and in M2, especially the relationships specified in OPM). Similarity in modeling across the layers reduces the “impedance mismatch,” which arises from ambiguous, poorly defined translations between the two levels of abstraction [23]. In particular, OPM/MAS utilizes the layers M2 and M1.5 as guidelines for developing a MAS and for validating the application model in layer M1. The constraints defined within these two layers can be enforced on a specification at the M1 layer. In particular, OPM/MAS aims at modeling the multi-agent system domain as a regular OPM application. As we use the same modeling language for both the domain and the application models, no mental transformation is required while moving between the intermediate

metamodel and the application model. This paper focuses on modeling within the M1.5 and M1 layers and on how to traverse them.

The OPM support for a multi-layered approach is enabled by two additional properties with which the OPM metamodel was extended: a multiplicity indicator and a classifier, which are similar to stereotypes in UML. The *multiplicity indicator*, defined and used within the M1.5 layer, states the number of elements that can appear within an application model. The *classifier* of a thing states the function of the OPM element. The classifier is used within the M1 layer in the application model, associating application elements with their corresponding agent-based system model elements, which appear in the M1.5 layer. The multiplicity indicator and classifier properties are attached to the Thing (Object or Process) class within the OPM metamodel [48]. The model residing in the intermediate metamodel layer (M1.5) describes the abstract syntax of the MAS model. It also describes the semantics of the MAS modeling language inheriting the OPM semantics, such as the execution order of processes.

Based on the clear distinction of OPM between objects and processes, which complement each other in modeling any system, the set of MAS building blocks and their attributes, presented in Section 2, was divided into two groups. The first group contains structural, declarative building blocks and attributes, whereas the second group contains building blocks and characteristics with behavioral, dynamic nature. The building blocks in the first group are modeled as OPM objects. These include organization, society, platform, role, user, protocol, environment, message, agent name, communication act, mobility type, mobility location, mobility priority, and mobility result. The building blocks in the second, behavioral, dynamic group, which include agent, task, messaging, and mobilizing, are modeled as OPM processes. Transformation of objects can be carried out by agents, tasks, messaging, or mobilizing. Therefore, the latter are classified as OPM processes. As noted, the model elements in both M1.5 and M1 layers refer to classes, not to instances. We also noted that we do not imply that the set defined above is the ultimate set of building blocks for MAS applications, but rather that the way we developed it and the considerations and constraints we have taken in account could be reused as a reasonable mechanism for specifying such a set.

Having introduced the general architecture of OPM/MAS, we next describe the intermediate metamodel of OPM/MAS, which enables specifying the building block to be used in MAS applications. The OPM/MAS specification employs the OPM modeling language syntax and semantics and the building blocks defined in Section 2. We emphasize that modeling at this intermediate metamodel level should be done by MAS experts, who are familiar with the MAS infrastructure and technology within an organization and can map the model elements to implementation artifacts. The developers of MAS applications can then use this metamodel for designing and modeling actual MAS applications.

Figure 4 is the system diagram (SD)—the top level OPM intermediate metamodel for MAS. This metamodel defines the relationships between the different MAS building blocks both structurally and behaviorally. For example, an **Organization** consists of **Roles** and exhibits **Agents**. A **Society** exhibits **Organizations** and **Agents**. Note that the "m" symbol, which appears in each of the OPM elements, is the multiplicity indicator, indicating here zero or more. Further explanations, similar to the explanations provided regarding the **Organization** and **Society** objects are provided by the associated OPL script in Figure 4.

In the OPD in Figure 6, **Agent** is in-zoomed and, as in any OPM model, this is specified both graphically and textually. For example, an **Agent** consists of **Task**, **Messaging**, and **Mobilizing** processes, and it exhibits (optionally many) **Objects**, **Mobility Locations**, **Mobility Priorities**, **Mobility Results**, **Mobility Types**, **Agent Names**, and **Communication Acts**. Note that the agent characteristics and parts are not identical in the OPDs of Figure 5 and Figure 6. This is so because in order to keep each OPD simple, OPM enables selective refinement of things. For example, in Figure 4, the **Agent** process does not appear to consist of a **Messaging** process, while in Figure 6 it does. The complete specification is the union of details within the entire OPD set.

The multiplicity indicators which appear on the links (such as "m" for many) denote the cardinality of the relationships between the building blocks. A value of the multiplicity on either side of a link can be one of the following: zero or one (0..1), exactly one (the default, no symbol), at least one (1..m), many (m), or any user-defined range. For instance, a **Task** can interact with many **Objects** (or none) and an **Object** can interact with at least one **Task**.

Since the notion of a protocol is essential in any agent-based system, it is elaborated in Figure 7 as an example of the OPM model of a MAS building block in layer M1.5. In OPM/MAS, a protocol is not restricted to some agent or role, but is rather modeled as an independent part of the multi-agent system. The OPD which specifies the protocol in Figure 7 includes the protocol itself (possibly as part of one or more other protocols, enabling protocol nesting), message groups (which enable consolidation of several messages), and communication acts, which can be related to each other, as well as to message groups.

Modeling multi-agent systems in M1.5 is a design choice. A designer may choose to model the MAS domain differently based on her perspective over the MAS domain and incorporate, for example, the Belief, Desire, and Intention (BDI) notion. This exemplifies a major advantage of OPM/MAS: it enables flexibility in defining the basic notions and elements of the MAS domain. The metamodel at the intermediate M1.5 level presented in Figures 4-6 will be followed throughout the paper.

eliminates the need to reinvent them from scratch for each system. The other purpose of the intermediate metamodel is to enforce correct usage of the MAS building blocks and the relationships among them.

The linkage between a thing (object or process) in the metamodel and its "incarnation" in the model is done via the classifier property, the name of the thing from the intermediate metamodel recorded in the upper left corner of the object's box or process' ellipse in the model. For example, OrchardBot in Figure 8 is classified as an agent.

5. OPM/MAS MODELING DEMONSTRATED

In this section we demonstrate the use of OPM/MAS intermediate metamodel within the application models through three case studies. The first case study—the Fruit Market—concerns a supply chain process in a merchandise market; it makes use of the core elements of OPM/MAS and demonstrates the relationship between the MAS intermediate metamodel and the model. The second case study—the Contractor Market—models a contractors' network and demonstrates the communication aspects within a multi-agent system. Finally, the third case study—the Technical Report Searcher—models a report search system and demonstrates the capabilities of OPM/MAS in modeling mobility aspects.

5.1. The Fruit Market: Linking the model and metamodel levels

The Fruit Market case study exemplifies the use of the MAS intermediate metamodel and demonstrates the core concepts of OPM/MAS and the benefit of its single, unified model. The Fruit Market is a marketplace application whose objective is to enable the market participants to electronically trade fruits over a network via agents as part of a supply chain process [10]. The Fruit Market consists of a market manager, an orchard owner, a fruit producer, and a supermarket chain, all interested in selling their products at the best possible price using an auction mechanism.

The specification of the Fruit Market system is based on the MAS intermediate metamodel. As noted, the classifiers recorded at the top-left corner of things (objects or processes) provide the links between the intermediate metamodel and the application model. For example, the object **Functional Host** in Figure 8 has the *Platform* classifier, which means that **Functional Host** in the model corresponds to a *Platform* in the intermediate metamodel and therefore it must follow the rules, expressed as patterns of links, determined for *Platform* in the intermediate metamodel in Figure 5.

The OPM/MAS model in Figure 8 demonstrates the dual purpose of the intermediate metamodel. One is the provisioning of guidelines for how to model an agent-based system and the other is the validation of the resulting artifacts against the intermediate metamodel. These guidelines prevent a designer of a new agent-based application from "reinventing the wheel" and instead use predefined building blocks from the intermediate metamodel and relationships

among them. Validation is to be carried out after specifying the system-to-be; the application model can be validated against the intermediate meta-model. For example, the **Fruit Market** organization consists of the **Trader**, **Manager** and **Organizer** roles. This is in line with the OPM/MAS intermediate metamodel in M1.5, which specifies in Figure 4 that *Organization* consists of at least one *Role*. Furthermore, the **Fruit Market**, classified as *Organization*, exhibits the agents **OrchardBot**, **SupplyBot**, **ShopBot** and **Broker**. This too conforms to the OPM/MAS intermediate metamodel, which specifies that *Organization* exhibits at least one *Agent*.

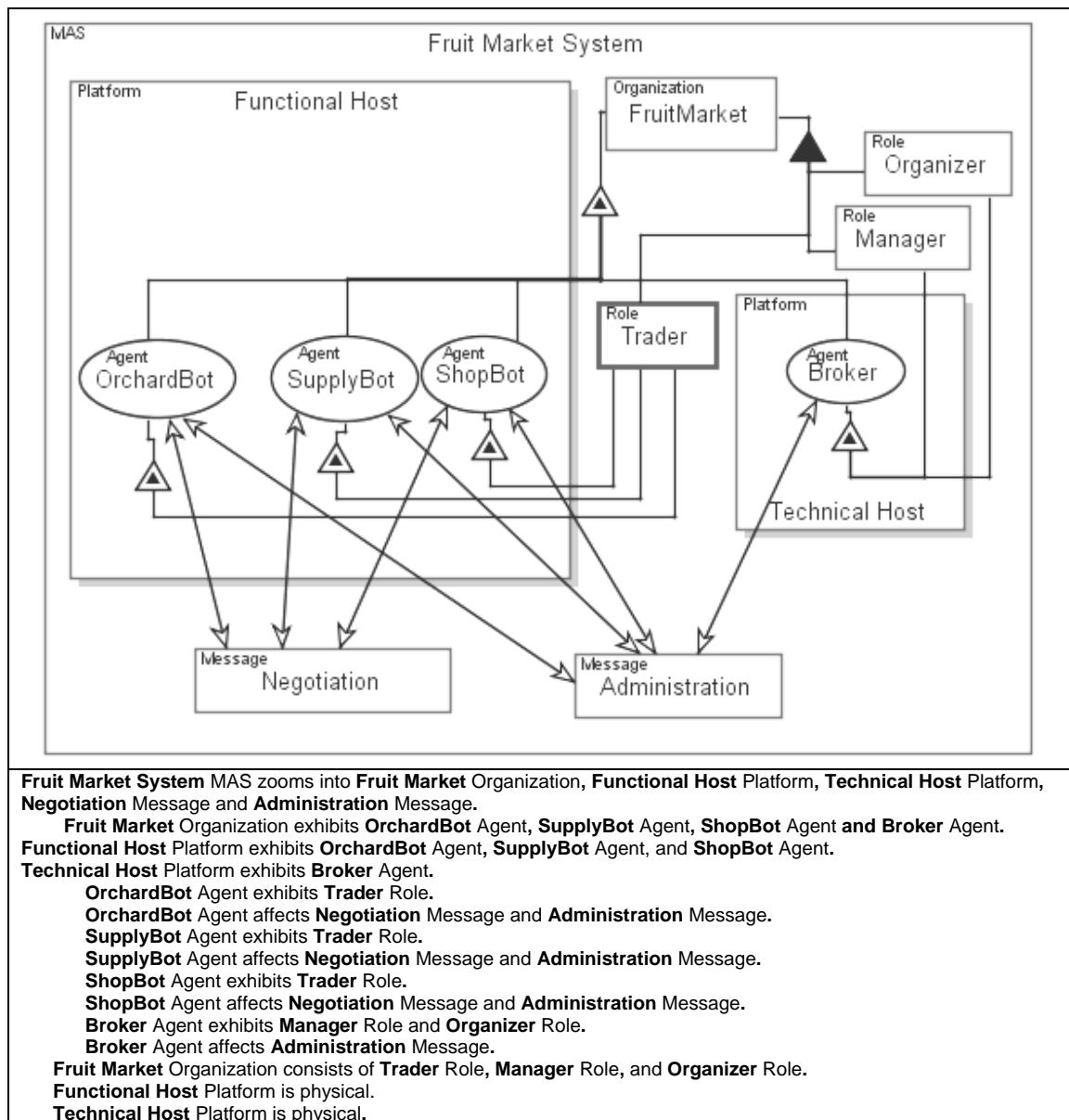


Figure 8. The Fruit Market system – System Diagram

The system architecture is comprised of two hosts, **Functional Host** and **Technical Host**, which are modeled in M1 (Figure 8) as *Platform*. The metamodel stipulates that *Platform* optionally exhibits many *Agents*. Indeed, the **Functional**

Host, which is a *Platform*, exhibits three *Agents*, while the **Technical Host**, which is another *Platform*, exhibits one *Agent*.

In the OPD of Figure 9, the **Trader Role** is unfolded, exposing four tasks. One of them, the **Buying Task**, zooms into three lower-level tasks: **Waiting**, **Needs Analyzing**, and **Bidding**. As in any OPD, the execution order of these three tasks is determined by their position along the vertical axis, with time flowing from top to bottom. Hence **Waiting** is executed first, followed by **Needs Analyzing**, and finally by **Bidding**. In this case, a new instance of the **Buying Task** is triggered in case **Buying Need** results with the state of **non-existent**. This is specified using the event link of OPM, which can be used to specify iterations. An example of iteration is presented in Section 5.3 within the Technical Report Searcher case study (e.g., Figure 15).

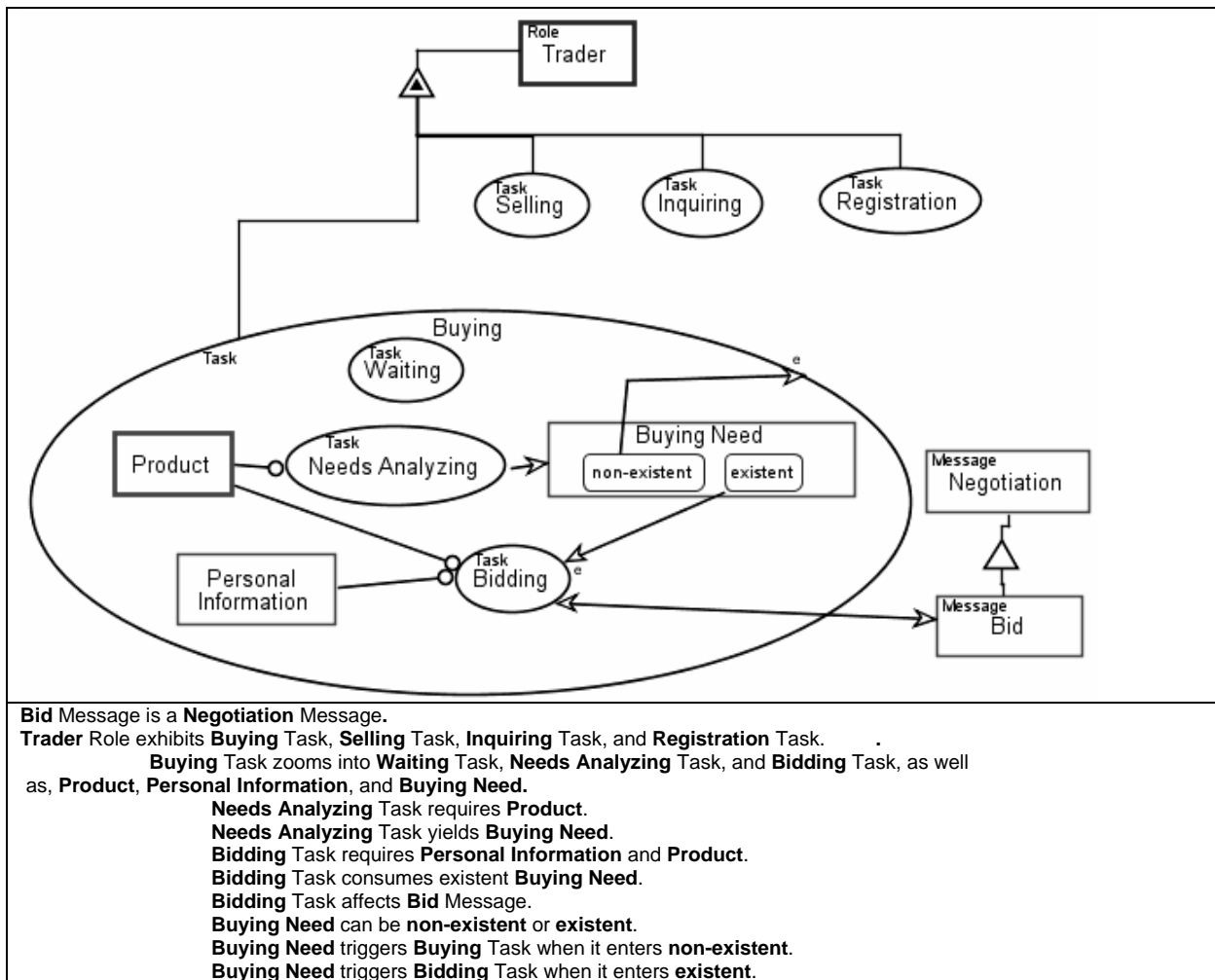


Figure 9. The Fruit Market system – Trader Role unfolded

5.2. The Contractor Market: modeling communication aspects

The Contractor Market is aimed at providing a framework for creating subcontracting relationships. Contractors can search for subcontractors that can perform specific tasks and make their choices based on financial considerations. This case study emphasizes modeling of communication aspects within a MAS. The first step in modeling of communication aspects in OPM/MAS is to specify the protocols, which determine communication patterns.

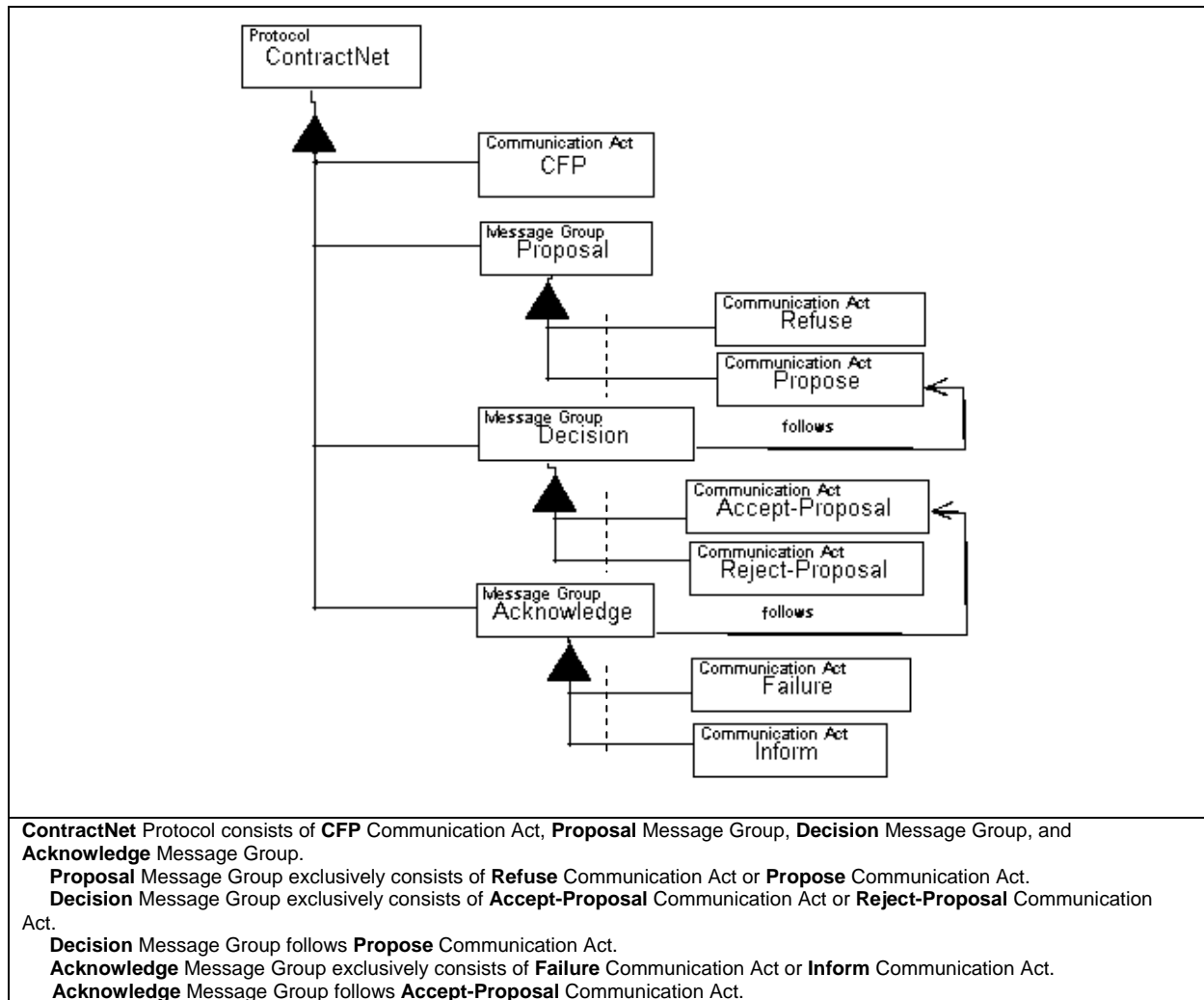


Figure 10. OPM/MAS model of the Contract Net protocol

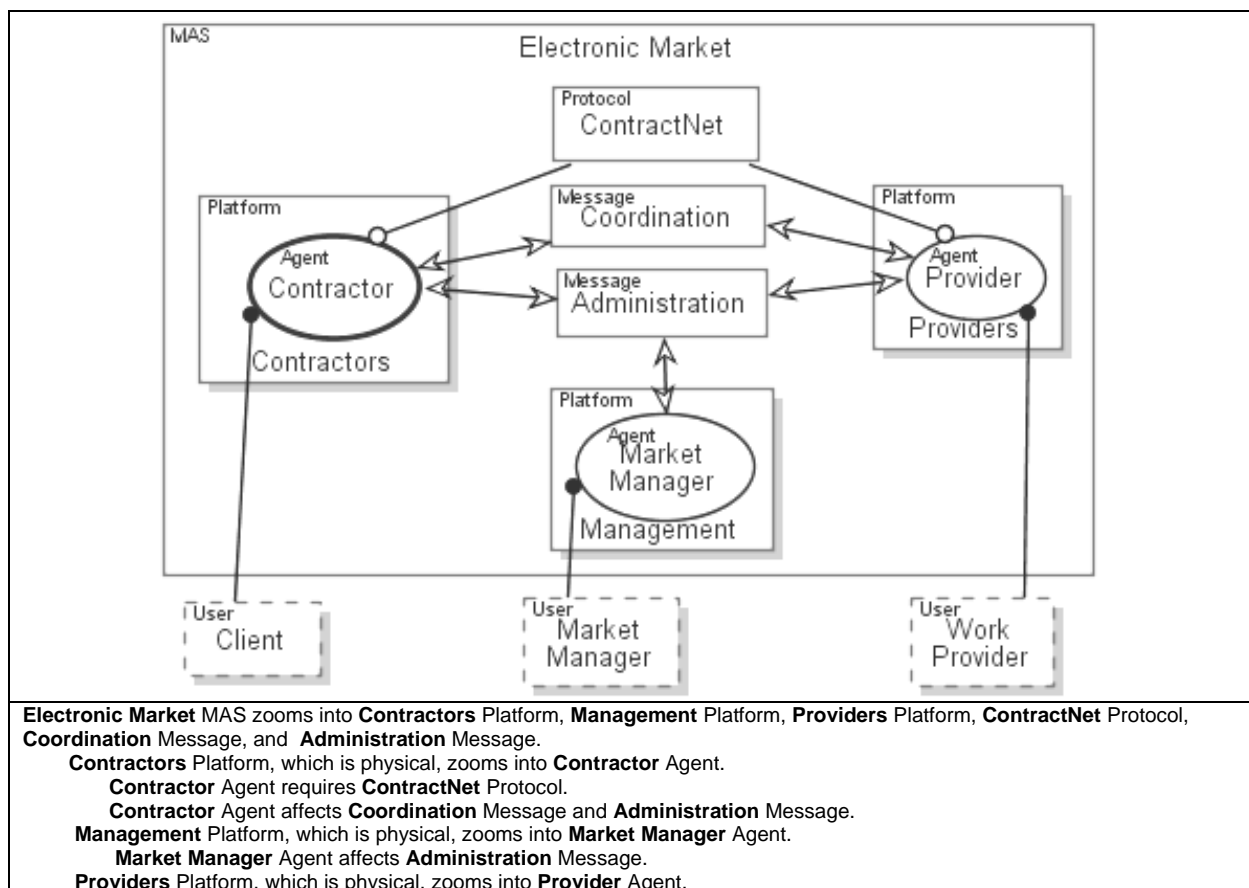
The protocol model used here is FIPA's Contract Net protocol [28]. The OPM/MAS model in Figure 10 presents the order of the message types, as determined by their vertical positions and their dependencies: **CFP** Communication Act, **Proposal** Message Group, **Decision** Message Group, and **Acknowledge** Message Group. The dashed line within each message group indicates a XOR relationship (of OPM) between the communication acts in that message group, implying that exactly one type of communication act can participate in an instance of the protocol. The unidirectional structural relations tagged **follows** between the **Decision** Message Group and the **Propose** Communication Act and between the **Acknowledge** Message Group and the **Accept-Proposal** Communication Act indicate the dependencies within the

protocol. For example, a message from the **Decision Message Group** can participate in an instance of the protocol only if it follows the **Propose Communication Act**. In the case of **Refuse Communication Act** and **Reject-Proposal Communication Act**, the protocol specifies that the communication ends.

Modeling the protocol is the first stage in specifying the system's communication. In the next stage, a designer should weave the protocol further into the agent behavioral specification. As the intermediate OPM/MAS metamodel suggests, the messages are part of the system flow. Hence, when a (receive or send) message is required, a messaging process is added to the flow of the agent or the task.

Figure 11 depicts the System Diagram (top-level OPD) of the **Electronic Market**—a MAS-based electronic market, which manages interactions among contractors. Upon receiving a task specification, a contractor looks for the appropriate agents and sends them a Call for Proposals (**CFP**) using FIPA's Contract Net protocol depicted in Figure 10.

The system consists of three agents: **Contractor**, **Provider**, and **Market Manager**, each residing on a different platform. The **Contractor** and the **Provider** communicate via the **ContractNet Protocol**, as indicated by the instrument links between **ContractNet Protocol** and each one of these two agents.

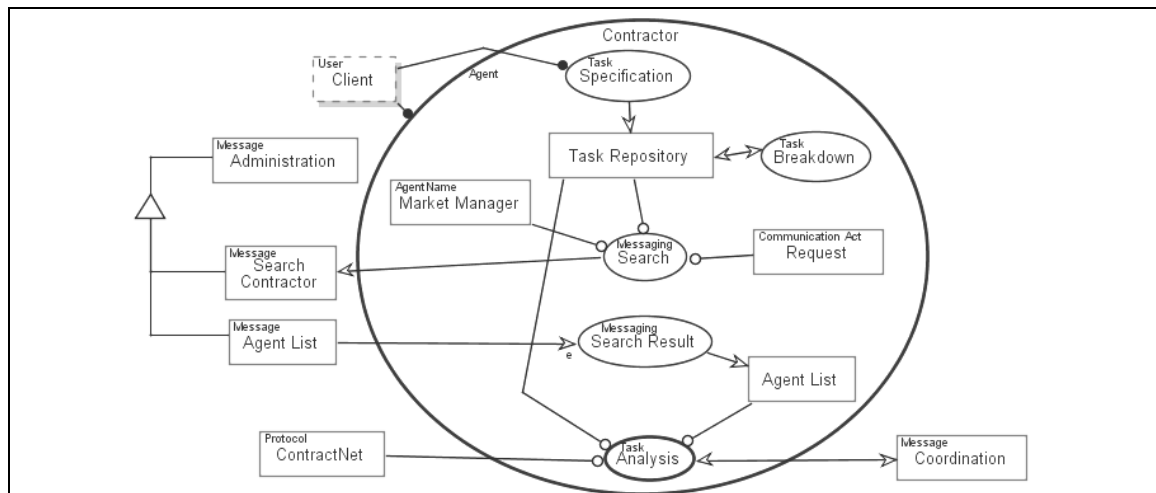


Provider Agent requires **ContractNet** Protocol.
Provider Agent affects **Coordination** Message and **Administration** Message.
Client User, which is environmental and physical, handles **Contractor** Agent.
Market Manager User, which is environmental and physical, handles **Market Manager** Agent.
Work Provider User, which is environmental and physical, handles **Provider Agent**.

Figure 11. The Electronic Market Multi-Agent System – System Diagram

In Figure 12, the **Contractor Agent** is in-zoomed, showing its tasks as sub processes. These start with the **Specification Task**, followed by messaging processes aimed at searching for possible subcontractors. Then, the **Analysis Task**, which is elaborated in Figure 13, takes place. Here, a specific agent is selected and then the **ContractNet** protocol takes place, as specified in the OPD of Figure 10. Specifying the communication aspects within the agent, each messaging process may be associated with an incoming/outgoing message, a communication act, or a protocol. A protocol associated with a messaging process indicates that the message must follow the protocol rules, i.e., the order of messages that the protocol specifies.

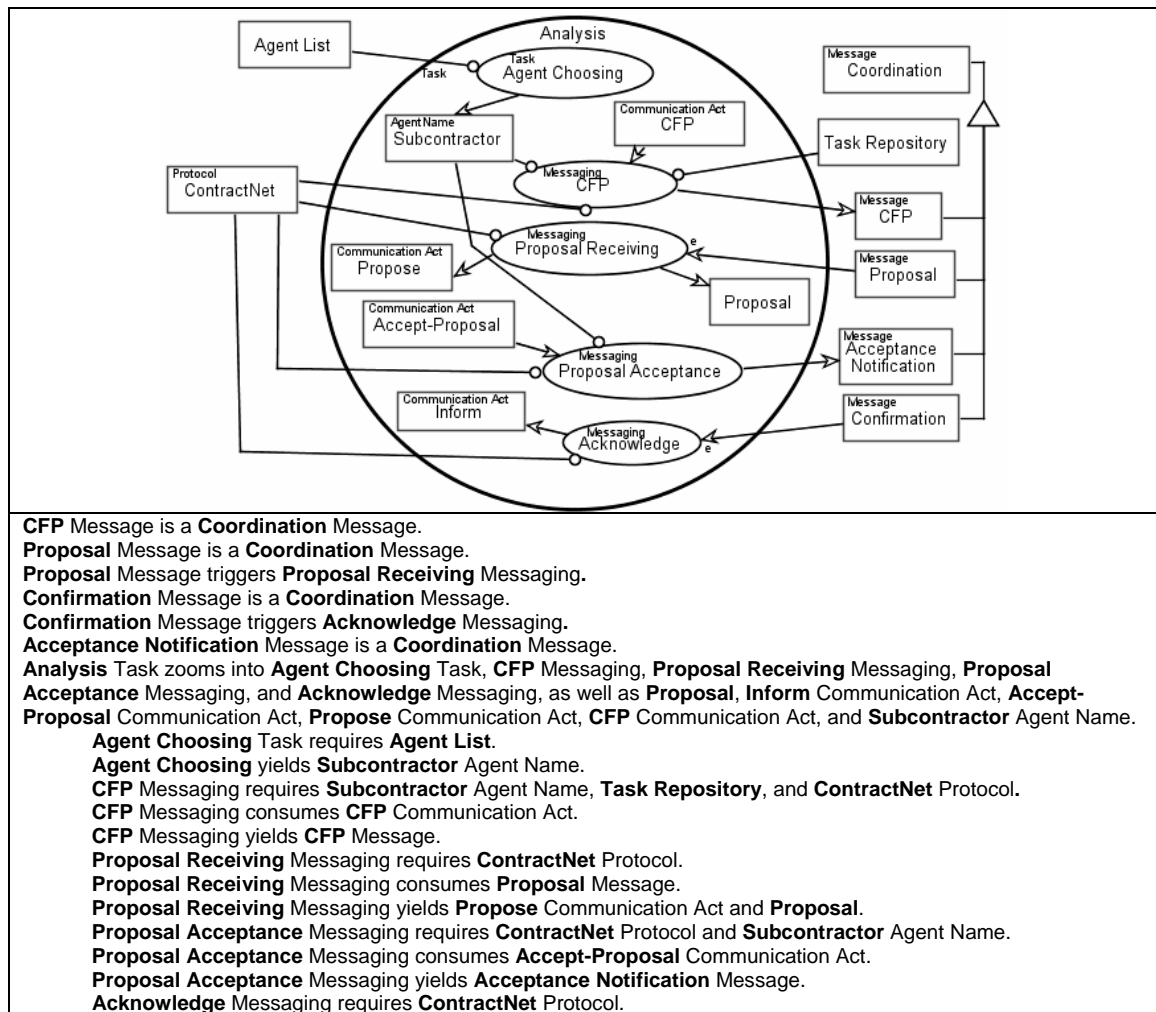
In this case study the same model is used for specifying the protocols and other system aspects. The way according to which the protocol is integrated into the system specification via the messaging process is also demonstrated. This model enables the designer to formally specify the communication aspects. As in the previous examples, this formality allows the designer to use the intermediate protocol metamodel for verification of the correct usage of the protocol within the system model.



Client User, which is environmental and physical, handles **Specification** Task and **Contractor** Agent.
Search Contractor Message is an **Administration** Message.
Agent List Message is an **Administration** Message.
Agent List Message triggers **Search Result** Messaging.
Contractor Agent zooms into **Specification** Task, **Breakdown** Task, **Search** Messaging, **Search Result** Messaging, and **Analysis** Task, as well as **Agent List**, **Market Manager** Agent Name, **Request** Communication Act, and **Task Repository**.
Specification Task yields **Task Repository**.
Breakdown task affects **Task Repository**.
Search Messaging requires **Request** Communication Act, **Market Manager** Agent Name, and **Task Repository**.
Search Messaging yields **Search Contractor** Message.
Search Result Messaging consumes **Agent List** Message.
Search Result Messaging yields **Agent List**.
Analysis Task requires **ContractNet** Protocol, **Agent List**, and **Task Repository**.
Analysis Task affects **Coordination** Message.

Figure 12. The Contractor Agent in-zoomed

In modeling communication aspects, OPM's expressiveness is of the same strength as other modeling languages in terms of message definition and protocol specification. However, the way protocols are defined in OPM/MAS enables its use in various agent-role combinations without the need to redefine them, allowing for reusability, which is a problematic issue in most MAS modeling languages. The single-view model also enables weaving the protocol messages into the entire system specification, providing for analysis of the agent behavior, a capability which is limited or nonexistent in other MAS modeling languages. OPM/MAS also provides a validation algorithm, which checks whether the specification of the messages within the agent functionality follows the relevant associated protocol. This issue is not handled by other MAS modeling languages [57]. However, the definition of protocols might suffer from accessibility problems, especially when compared to similar definitions performed using the AUML interaction diagram. That is, in OPM/MAS it might be difficult to understand the message flow of a specific interaction, as it focuses on the single agent's behavior and not on the interaction.

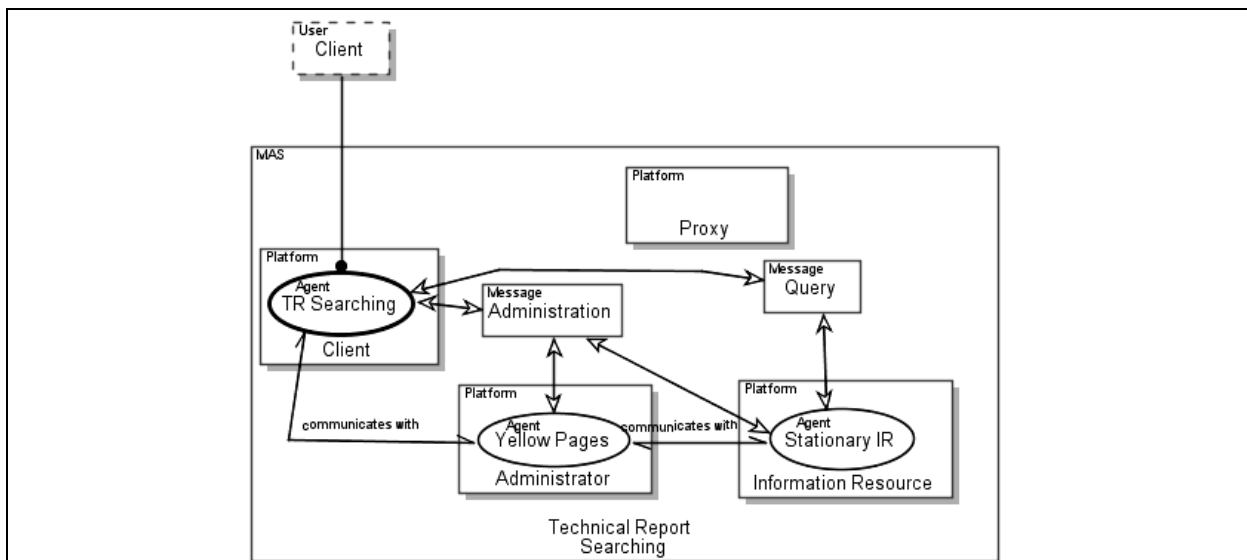


Acknowledge Messaging consumes **Confirmation** Message.
Acknowledge Messaging yields **Inform** Communication Act.

Figure 13. The Analysis Task in-zoomed

5.3. The Technical Report Searcher: Modeling agent mobility

As a final example, an application of OPM/MAS for modeling mobile agents and distributed information retrieval via the technical report (TR) searcher case study [31] is presented. In this case study, technical reports are distributed across several machines, each executing a stationary information retrieval agent. When these agents are executed, they register with a virtual yellow pages agent. In search for technical reports, a client agent queries the yellow pages agent for the location of the stationary information retrieval agents. It then sends its "child" agents to these locations. These child agents interact with the stationary agents, which in turn reply to the child agents with the search results. In addition, the client agent checks the network quality and determines whether it requires migrating to a proxy site in order to communicate with the machines hosting the stationary agents. Figure 14 presents the system diagram of the Technical Report Searcher system. It consists of four platform types: **Client Platform**, which hosts the **TR Searching Agent**; **Administrator Platform**, which hosts the **Yellow Pages Agent**; the **Information Resource Platform**, which hosts the **Stationary IR (Information Retrieval) Agents**; and the **Proxy Platform**, which is capable of hosting the **TR Searching Agent** in case of faulty communication between the **Client Platform** and the **Information Resource Platform**. The OPD in Figure 14 also describes the communication paths, which are the logical routes among the agents, and the messages.



Technical Report Searching MAS zooms into **Client Platform**, **Proxy Platform**, **Administrator Platform**, **Information Resource Platform**, **Administration** Message, and **Query** Message.

Client Platform, which is physical, zooms into **TR Searching** Agent.

TR Searching Agent affects **Administration** Message and **Query** Message.

Proxy Platform is physical.

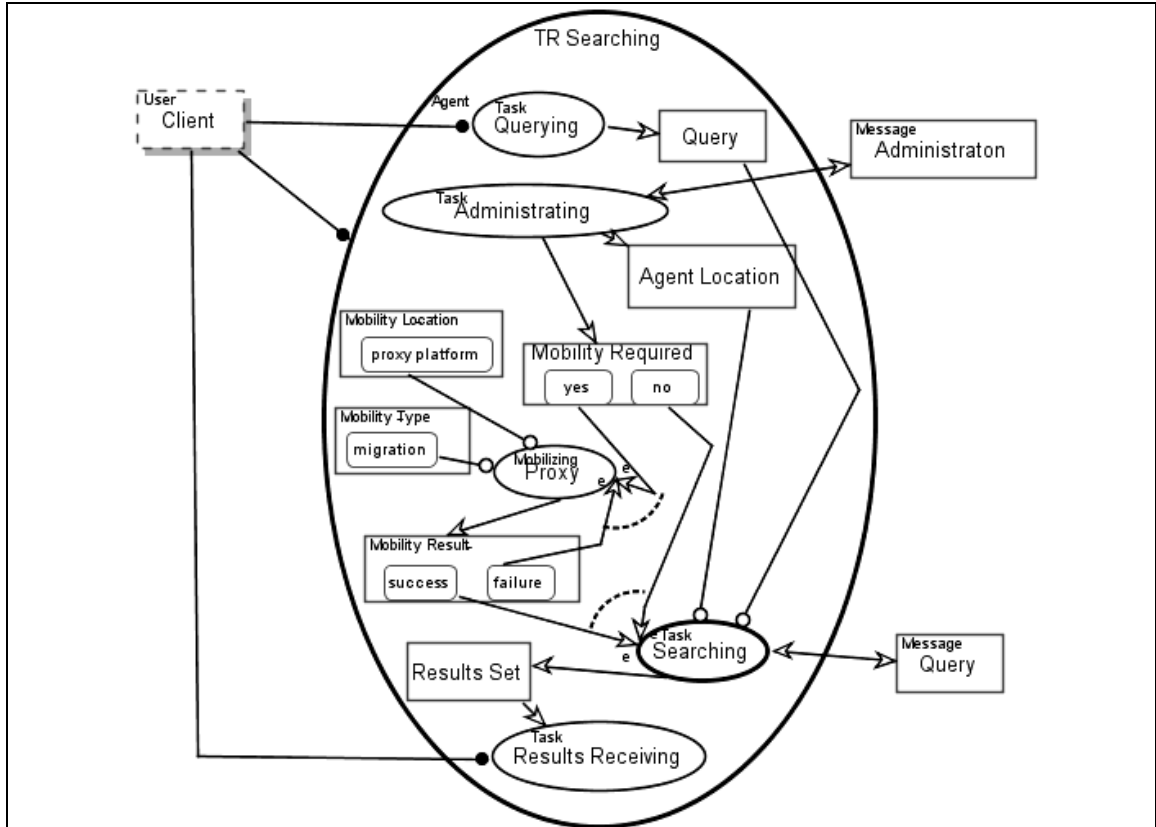
Administrator, which is physical, zooms into **Yellow Pages** Agent.

Yellow Pages Agent affects **Administration** Message.

Information Resource Platform, which is physical, zooms into **Stationary IR** Agent.

Stationary IR Agent affects **Query** Message and **Administration** Message.
Yellow Pages Agent communicates with **Stationary IR** Agent.
Yellow Pages Agent communicates with **TR Searching** Agent.
Stationary IR Agent communicates with **Yellow Pages** Agent.
TR Searching Agent communicates with **Yellow Pages** Agent.
Client User, which is environmental and physical, handles **TR Searching** Agent.

Figure 14. Technical Report Searching System – System Diagram



Client User, which is environmental and physical, handles **TR Searching** Agent, **Querying** Task, and **Results Receiving** Task.

TR Searching Agent zooms into **Querying** Task, **Administrating** Task, **Proxy Mobilizing**, **Searching** Task, and **Results Receiving** Task, as well as **Results Set**, **Agent Location**, **Query**, **Mobility Required**, **Mobility Type** Object, **Mobility Location** Object, and **Mobility Result** Object.

Querying Task yields **Query**.

Administrating Task affects **Administration** Message.

Administrating Task yields **Agent Location** and **Mobility Required**.

Mobility Required can be **yes** or **no**.

Mobility Required triggers **Proxy Mobilizing** when it enters **yes**.

Mobility Required triggers **Searching** Task when it enters **no**.

Proxy Mobilizing requires **migration** **Mobility Type** Object and **proxy platform** **Mobility Location** Object.

Proxy Mobilizing consumes **failure** **Mobility Result** Object.

Proxy Mobilizing yields **Mobility Result** Object.

Mobility Result Object can be **success** or **failure**.

Mobility Result Object triggers **Searching** Task when it enters **success**.

Mobility Result Object triggers **Proxy Mobilizing** when it enters **failure**.

Searching Task requires **Agent Location** and **Query**.

Searching Task affects **Query** Message.

Searching Task consumes **success** **Mobility Result** Object.

Searching Task yields **Results Set**.

Results Receiving Task consumes **Results Set**.

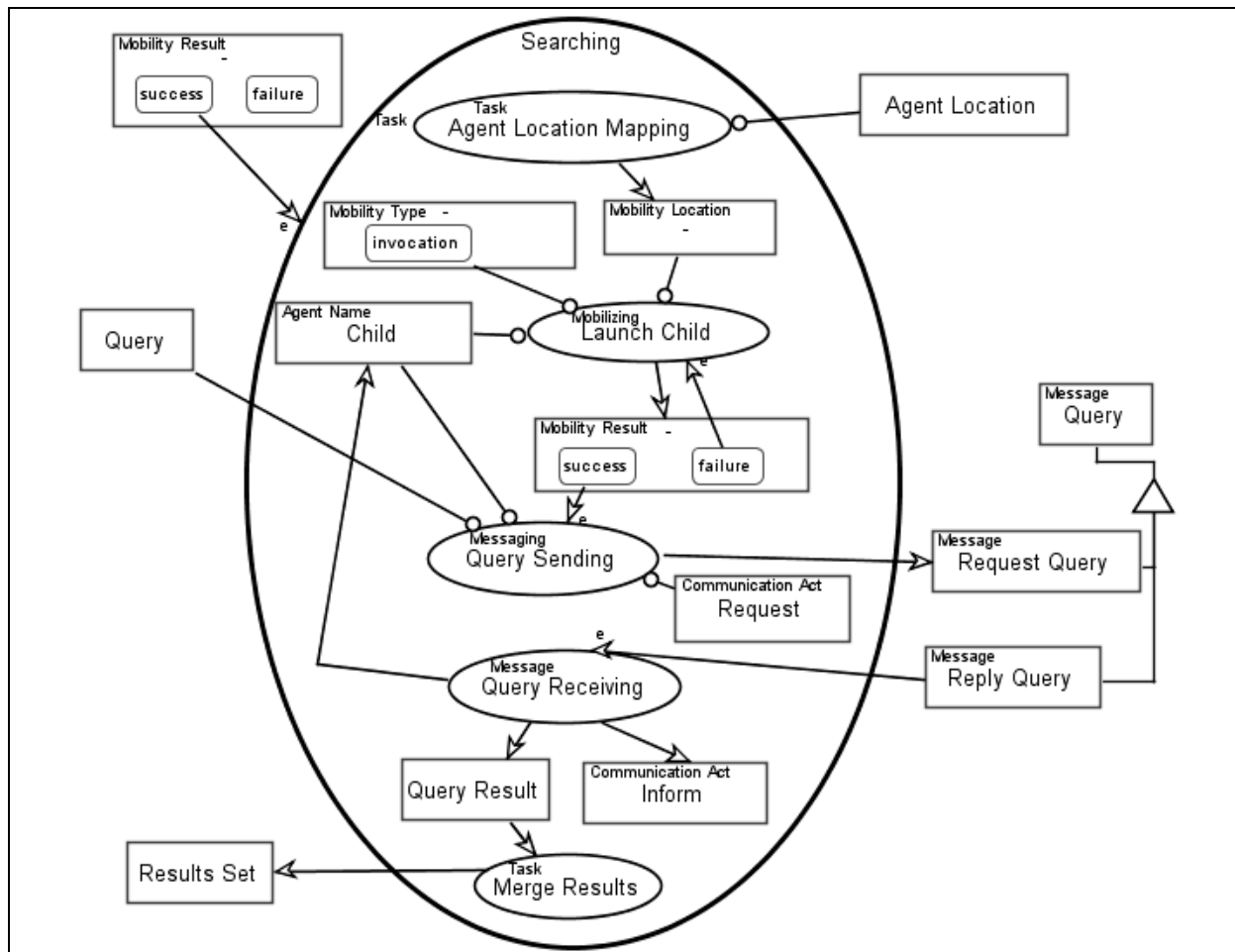
Figure 15. Technical Report Searching System – TR Searching Agent in-zoomed

In Figure 15, the **TR Searching Agent** is in-zoomed. The agent is activated by the **Client User**, as shown by the agent link from the object **Client User** (a human) to the process **TR Searching Agent**. The **TR Searching Agent** performs its

tasks sequentially, as determined by the vertical order within the OPD. The **Querying Task** accepts the user's input and yields a **Query** object. The **Administrating Task** follows the **Querying Task** and yields an object indicating whether mobility is required and an object representing the required *Agent Location*. If mobility is required, the **Proxy Mobilizing** process occurs and causes the **TR Searching Agent** to migrate to the **Proxy Platform**. If the **Proxy Mobilizing** process fails, then it is triggered again. This is indicated by the “e”, for "event", attached to the arrowhead of the link connecting the failure state within the *Mobility Result* object and the **Proxy Mobilizing** process. When the mobility result is a success (or if it was not required in the first place), the **Searching Task** is performed, followed by the **Results Receiving Task**.

In Figure 16, the **Searching Task** from the OPD of Figure 15 is specified. It begins with the **Agent Location Mapping Task**, which determines *Mobility Location*, in which the **Child Agent** has to be invoked. This task is followed by launching the **Child Agent** in the appropriate platform. In case the invocation fails, the agent iteratively tries to re-launch the **Child Agent** until it succeeds. When the **Child Agent** is running, the **TR Searching Agent** sends a **Request Query Message** and waits for a reply. Upon receiving the query results, **Result Query Message**, the results received from its child agents contained in the message are merged to obtain the **Results Set**, which the **Client User** ultimately receives (see Figure 15).

This case study demonstrates the use of OPM/MAS for specifying mobile agents and how agent migration and invocation is modeled within the agent activity. Further discussion on the OPM/MAS support for mobility aspects appears in [62].



Mobility Result Object can be **success** or **failure**.
 Mobility Result Object triggers **Searching** Task when it enters **success**.
Request Query Message is a **Query** Message.
Reply Query Message is a **Query** Message.
Reply Query Message triggers **Query Receiving** Messaging.
Searching Task consumes **success** Mobility Result Object.
Searching Task zooms into **Agent Location Mapping** Task, **Launch Child** Mobilizing, **Query Sending** Messaging, **Query Receiving** Messaging, and **Merge Results** Task, as well as **Query Result**, Mobility Type Object, Mobility Location Object, **Child** Agent Name, Mobility Result Object, **Request** Communication Act, and **Inform** Communication Act.
Agent Location Mapping Task requires **Agent Location**.
Agent Location Mapping yields Mobility Location Object.
Launch Child Mobilizing requires **invocation** Mobility Type Object, Mobility Location Object, and **Child** Agent Name.
Launch Child Mobilizing consumes **failure** Mobility Result Object.
Launch Child Mobilizing yields Mobility Result Object.
 Mobility Result Object can be **success** or **failure**.
 Mobility Result Object triggers **Query Sending** Messaging when it enters **success**.
 Mobility Result Object triggers **Launch Child** Mobilizing when it enters **failure**.
Query Sending Messaging requires **Child** Agent Name, **Query**, and **Request** Communication Act.
Query Sending Messaging consumes **success** Mobility Result Object.
Query Sending Messaging yields **Request Query** Message.
Query Receiving Messaging consumes **Reply Query** Message.
Query Receiving Messaging yields **Inform** Communication Act, **Query Result**, and **Child** Agent Name.
Merge Results Task consumes **Query Result**.
Merge Results Task yields **Results Set**.

Figure 16. Technical Report Searching System – Searching Task in-zoomed

6. EVALUATING OPM/MAS

We have evaluated OPM/MAS using a variety of software engineering evaluation methods. These include a feature analysis, a survey, an independent evaluation framework, and an empirical comparison. In this section we briefly present these evaluations and discuss them. The details of each evaluation appear in other publications, cited below.

6.1. Feature-Based Analysis

The feature-based analysis we conducted is based on an evaluation framework we developed [58]. The framework examines aspects concerning the concepts and properties of the method, the notation and modeling techniques provided by the method, the development process supported by the method, and the method pragmatics. Among other criteria, this framework includes accessibility and expressiveness. The feature-based analysis of OPM/MAS has established that OPM/MAS exhibits desired traits expected from a “good” MAS modeling language. OPM/MAS was found deficient with respect to the development process, as it does not refer to the various stages within it. However, this is a methodology aspect, not a language one. Elaboration of that analysis can be found in [56, 60].

6.2. A Survey

We have checked the level of understanding of various agent-oriented methods by novices. Specifically, we surveyed their rankings of properties, in particular agent-based system characterization and software engineering properties of these methods. The survey was conducted as part of an AOSE course in which students learned the notion of agent-based systems in general and AOSE aspects in particular. OPM/MAS was one of the methods that were evaluated. The responses indicated that OPM/MAS supports agent-based system characterization to a satisfactory level and that it supports software engineering properties to a good level. OPM/MAS ranked higher than, GAIA, MaSE, and Tropos, which were also evaluated in that study. However, the same study also evaluated OPM/MAS low in addressing communication aspects. Further details can be found in [56].

6.3. Evaluation with an External Framework

We evaluated OPM/MAS with an objective evaluation framework, described in [13], which examines five categories: (1) the development process, (2) the models, (3) the supported concepts, (4) additional modeling features, and (5) the methodology documentation. We found that OPM/MAS addresses well most of the framework criteria, but its documentation is insufficient with respect to the development process. Yet, as OPM/MAS is based on OPM, its underlying notation and semantics are described in great detail and precision in [20]. Using this framework to compare OPM/MAS with the above mentioned methodologies [14], we found that OPM/MAS achieved better results. Further details of this analysis can be found in [56].

6.4. Empirical Evaluation

To gain a quantitative empirical evaluation of OPM/MAS, we conducted a controlled experiment that compared MaSE with OPM/MAS. The experiment, which examined the comprehension and construction of models using these two methods, took place as part of an examination in an agent-oriented software engineering course. The students achieved similar results when using both methods. Yet, some statistically significant differences were discovered. Specifically, OPM/MAS was found to be more helpful than MaSE in understanding the entire system and in specifying interactions with the environment. Details of that study can be found in [61].

6.5. Additional Comparison

While various properties of modeling languages for MAS were evaluated using the aforementioned tools, flexibility, which is a major capability of OPM/MAS, did not receive adequate attention. For the sake of comparing flexibility we selected a representative set of MAS modeling languages to be compared to OPM/MAS, which included Gaia, MaSE, AUML, and Prometheus. Three reasons accounted for selecting these modeling languages: (1) they all originated from the software engineering domain, (2) they are aimed at specifying general purpose agent-based systems, and (3) they cover the major modeling terminology, notions, and notations within the area of agent-oriented modeling languages. Other methodologies and modeling languages adopt modeling aspects from these languages or deal with specific aspects of MAS.

AUML, Gaia, Prometheus, and MaSE lack with respect to flexibility, as they each have their own rigid metamodel and do not allow users to change it. Although any metamodel can be changed, these representative languages do not refer to such change, as their entire development process relies on their fixed, unchangeable metamodels. In OPM/MAS, an expert user can change the intermediate level metamodel in order to incorporate new agent-oriented building blocks or modify existing ones to provide for a particular agent-oriented abstraction or usage. This provides the modeler with the flexibility of tailoring an agent-oriented modeling language while maintaining the formal base definitions.

With respect to flexibility, OPM/MAS is similar to INGENIAS [30, 45, 46] in its meta-modeling-based approach. The latter leverages on Message/UML [7] and adds a process layer on top of the various meta-models to enable their synchronization. Indeed INGENIAS provides rich means for modeling MAS applications, yet it suffers from the model multiplicity problem, as it uses several views. Although the developers of INGENIAS state that their metamodels can be changed, they do not elaborate on this issue in their publications. In OPM/MAS the metamodel can be changed within the M1.5 layer. In INGENIAS the metamodel is specified using GOPRR [38] while the modeling is done using different notations. In OPM/MAS the intermediate metamodel and the model use the same OPM-based notation. Using the same

language for the metamodel and the model layers reduces the mental transformation efforts that users are required to exercise, increasing the likelihood that the language will be usable and adopted by users.

Indeed, other modeling language types, including those for modeling electronic institutions or belief-desire-intentions, might be suitable for various cases. For example, electronic institutions are suitable when the organization aspect should be specified and elaborated. The work of Islander [22] can be used for modeling systems such as the market case studies presented in this paper. The Belief-Desire-Intention (BDI) notion has also been used in multi-agent systems for representing knowledge and reasoning capabilities [39]. OPM/MAS can also be adapted for that purpose. However, in this paper we refer to the general notion of MAS modeling language.

OPM/MAS has been successfully applied to specify a large-scale multi-agent system, including a personal itinerary planning system whose goal was to assist a traveler in organizing her travel plans and activities, and the Guardian Angel system, whose goal was to improve the quality of healthcare services [55].

7. CONCLUSIONS

This paper presents OPM/MAS— an alternative modeling language for multi-agent systems. The novelty of this MAS modeling language is partially due to its three-layered approach. The three layers are the base modeling language layer (which is OPM), the intermediate metamodel layer, which defines the building blocks of MAS and their semantics, and the model layer, in which actual MAS are developed based on the two other layers.

In this study, a set of core MAS concepts serves as a basis for the intermediate metamodel level. As we have demonstrated, this model can specify a wide range of MAS applications. Because to its OPM foundations, OPM/MAS exhibits enhanced expressiveness and accessibility capabilities, which in turn facilitate its learning and usage. The three-layer approach provides OPM/MAS with the desired flexibility attribute, which is achieved by supporting the ability to introduce changes into the intermediate metamodel, while keeping the same base modeling language.

While maintaining its flexibility, the current OPM/MAS in specification, including the intermediate metamodel layer, addresses many desirable aspects of MAS applications. However, it is cumbersome with respect to the communication view of a MAS. This issue can be resolved in various ways. The first is to use the views capability of OPM and its straightforward implementation in OPCAT in order to filter out elements that are not directly related to the communication aspect, thus resulting in a message flow specification that was obscured or not apparent when the communication elements were integrated within the system's functional specification. The second solution (which does not preclude the first one) involves taking advantage of the simulation capability of OPM to track the message flow

within the modeled MAS. The third option of tackling limitations related to modeling the MAS communication aspects is to modify the intermediate metamodel to better support this aspect. This way, one can also support other combinations of MAS building blocks, such as those required for BDI.

OPM/MAS, which follows the model-driven approach, has been enhanced by transformation rules to JADE [64], providing for OPM/MAS model continuity [56], so OPM/MAS models can be further used for the implementation phase. We plan to continuously evaluate OPM/MAS and augment OPCAT [21], the OPM integrated systems engineering environment, with additional features to support OPM/MAS. At this stage, OPCAT supports the association of M1.5 elements to M1 elements and partial validation of a model in M1 with respect to the corresponding model in M1.5. The extension of OPCAT to support OPM/MAS will enable MAS metamodeling, enforce the linkage between the MAS metamodel and a MAS model, check the model compliance with the metamodel, and suggest changes where the model deviates from the metamodel.

In future research, we plan to explore OPM/MAS with various intermediate metamodels specifying various types of MAS applications, and examine whether OPM/MAS can be used with various agent methodologies, particularly Gaia, Tropos, and Prometheus. We believe that the ideas embodied in OPM/MAS can lead to an agreement on a unified modeling notation for the plethora of agent methodologies. We also plan to experiment with OPM/MAS in industrial settings.

REFERENCES

1. AgentLink, <http://www.agentlink.org/>, 2004
2. B. Bauer, J. P. Müller, and J. Odell, Agent UML: A Formalism for Specifying Multiagent Software Systems, *International Journal of Software Engineering and Knowledge Engineering* 11(3) (2001), 207-230.
3. C. Bernon, M. Cossentino, M.-P. Gleizes, P. Turci, and F. Zambonelli, A Study of some Multi-agent Meta-models, in: the 5th International Workshop on Agent-Oriented Software Engineering. *Lecture Notes in Computer Science* 3382, Springer Verlag, 2005, 62-77.
4. C. Bernon, M.-P. Gleizes, G. Picard, and P. Glize, The ADELFE methodology for an intranet system design, in: *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS)*, Toronto, Canada, 2002, P. Giorgini, Y. Lesperance, G. Wagner and E. Yu Eds.
5. F. M. T. Brazier, B. Dunin-Keplicz, N. R. Jennings, and J. Treur, DESIRE: Modeling Multi-Agent Systems in a Compositional Formal Framework, *International Journal of Cooperative Information Systems* 6 (1997), 67-94.
6. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, TROPOS: An Agent-Oriented Software Development Methodology. *Journal of Autonomous Agents and Multi-Agent Systems* 8(3) (2004) 203 - 236.

7. G. Caire, F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, J. Pavon, P. Kearney, J. Stark, and P. Massonet, Agent Oriented Analysis using MESSAGE/UML, in: Proceeding of the Second International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science 2222, Springer Verlag, 2001, 119-135.
8. J. Castro, P. Giorgini, S. Kethers, and J. Mylopoulos, A Requirements-Driven Methodology for Agent-Oriented Software, in: Agent-Oriented Methodologies, B. Henderson-Sellers and P. Giorgini, Eds, Idea Group, 2005.
9. R. Cervenka and I. Trencansky, The Agent Modeling Language – AML: A Comprehensive Approach to Modeling Multi-Agent Systems, Whitestein Series in Software Agent Technologies and Autonomic Computing, Springer Verlag, 2007
10. J. C. Collis and L. C. Lee, Building electronic marketplaces with the zeus agent toolkit, in: Proceedings of Agent Mediated Electronic Trading. Lecture Notes in Artificial Intelligence 1571, Springer Verlag, C. Sierra and P. Noriega, Eds., 1999.
11. M. Cossentino, C. Bernon, J. Pavòn, Modelling and Meta-modelling Issues in Agent Oriented Software Engineering: The AgentLink AOSE TFG Approach, http://www.pa.icar.cnr.it/cossentino/al3tf2/docs/aosetfg_report.pdf, 2005.
12. M. Cossentino, S. Gaglio, A. Garro, and V. Seidita, Method fragments for agent design methodologies: from standardization to research, International Journal of Agent-Oriented Software Engineering 1(1) (2007), 91-121.
13. P. Cuesta, A. Gómez, J. C. González, and F. J. Rodríguez, A Framework for Evaluation of Agent Oriented Methodologies, in: Proceedings of the Conference of the Spanish Association for Artificial Intelligence, 2003.
14. Cuesta P., Gómez A., González J. C., and Rodríguez F. J., “Evaluating Agent-Oriented Methodologies”, <http://ma.ei.uvigo.es/framework/>, 2003
15. S. A. DeLoach, M. F. Wood, and C. H. Sparkman, Multiagent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering 11(3) (2001), 231-258.
16. S.A. DeLoach, Multiagent Systems Engineering of Organization-based Multiagent Systems, in: Proceedings of the Forth International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05), Lecture Notes in Computer Science 3914, 2005, 109-125.
17. S.A. DeLoach, Developing a Multiagent Conference Management System Using the O-MaSE Process Framework, in: the Eighth International Workshop on Agent Oriented Software Engineering, 2007.
18. M. d'Inverno and M. Luck, Understanding Agent Systems (2nd Edition), Springer Verlag, 2003.
19. B. Dobing and J. Parsons, How UML is used, Communication of the ACM 49(5) (2006), 109-113.
20. D. Dori, Object-Process Methodology - A Holistic Systems Paradigm, Springer, Heidelberg, 2002.
21. D. Dori, I. Reinhartz-Beger, and A. Sturm, OPCAT – A Bimodal CASE Tool for Object-Process Based System Development, in: Proceeding of the Fifth International Conference On Enterprise Information Systems (ICEIS) 3, 2003, 286-291.
22. M., Esteva, D. de la Cruz, and C. Sierra, ISLANDER: an electronic institutions editor. In Proceedings of the First international Joint Conference on Autonomous Agents and Multiagent Systems: Part 3, 2002, 1045-1052.
23. J. Evermann and Y. Wand, Toward formalizing domain modeling semantics in language syntax, IEEE Transactions on Software Engineering 31(1) (2005), 21 – 37.
24. FIPA, Agent Management Support for Mobility Specification. <http://www.fipa.org/specs/fipa00005/index.html>, 1998..
25. FIPA, FIPA ACL Message Structure Specification. <http://www.fipa.org/specs/fipa00061/index.html>, 2002.
26. FIPA, www.fipa.org, 2003.

27. FIPA, FIPA Modeling Area: Environment. <http://www.auml.org/auml/documents/Environment-030412.pdf>, 2003.
28. FIPA, FIPA Contract Net Interaction Protocol Specification, <http://www.fipa.org/specs/fipa00029/>, 2003.
29. S. J. Franklin, Assessing the suitability of UML for Capturing and Communicating Systems Engineering Design Models, Vitech Corporation, http://www.vitechcorp.com/library/docs/SuitabilityOfUMLForSE_2002.pdf, 2002.
30. J. J. Gómez-Sanz and J. Pavón, Meta-modelling in Agent Oriented Software Engineering, in: Proceedings of the Eighth Iberoamerican Conference on Artificial Intelligence (IBERAMIA), 2002, 606-615.
31. R.S.Gray, G. Cybenko, D. Kotz, and D. Rus, Mobile Agents: Motivations and State of the Art, in: Handbook of Agent Technology, J. Bradshaw, Ed., 2001.
32. K. Hoa Dam and M. Winikoff, Comparing Agent-Oriented Methodologies, In Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information System (AOIS), Lecture Notes in Computer Science 3030, Springer Verlag., 2004, 78-93.
33. C. A. Iglesias, M. Garijo, and J.C. Gonzalez, A survey of agent-oriented methodologies, in: Proceedings of ATAL-98, Lecture Notes in Computer Science 1555, 1999, 317-330.
34. C. A. Iglesias, M. Garijo, J.C. Gonzalez, and J.R. Velasco, Analysis and Design of multiagent systems using MAS-CommonKADS, in: Proceedings of ATAL-97, Lecture Notes in Computer Science 1365, 1997, 313-328.
35. N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, B. Odgers, and L. J. Alty, Implementing a Business Process Management System using ADEPT: A Real-World Case Study, International Journal of Applied AI 14(5) (2000), 421-465.
36. N. R. Jennings, K. P. Sycara, and M. Wooldridge, A Roadmap of Agent Research and Development, Journal of Autonomous Agents and Multi-Agent Systems 1(1) (1998), 7-36.
37. J. Kabeli and P. Shoval, FOOM: functional- and object-oriented analysis & design of information systems - an integrated methodology, Journal of Database Management 12(1) (2001), 15-25.
38. S. Kelly, K. Lyytinen, and M. Rossi, METAEDIT+ A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment, Lecture Notes in Computer Science 1080, Springer Verlag., 1996.
39. D. Kinny, M. Georgeff, and A. Rao, A methodology and modelling technique for systems of BDI agents, in Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, 1996, 56-71.
40. P. Massonet, Y. Deville, and C. Neve, From AOSE Methodology to Agent Implementation, in: Proceedings the First International Joint Conference on Autonomous agents and Multiagent Systems, 2002, 27 – 34.
41. H. Nwana, D. Ndumu, L. Lee, and J. Collis, ZEUS: A Tool- Kit for Building Distributed Multi-Agent Systems, Applied Artificial Intelligence Journal 13(1) (1999), 129-186.
42. OMG, Meta Object Facility (MOF) 2.0, 2006.
43. OMG, UML 2.0 Superstructure, 2006.
44. L. Padgham and M. Winikoff, Prometheus: A Methodology for Developing Intelligent Agents: in: Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, 2002.
45. J. Pavón and J. J. Gómez-Sanz, Agent Oriented Software Engineering with INGENIAS, in: Proceedings Third International/Central and Eastern European Conference on Multi-Agent Systems (CEEMAS), Lecture Notes in Computer Science 2691, Springer Verlag, 2003, 394-403.

46. J. Pavón, J. Gómez-Sanz, and R. Fuentes, The INGENIAS methodology and tools, in: *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini, Eds., Idea Group Publishing, 2005, 236–276.
47. M. Peleg and D. Dori, The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods, *IEEE Transaction on Software Engineering* 26(8) (2000), 742-759.
48. I. Reinhartz-Berger, "Developing Web Applications with Object-Oriented Approaches and Object-Process Methodology", Ph.D. dissertation, <http://mis.hevra.haifa.ac.il/~iris/research/OPMwebThesis.pdf> , 2003.
49. I. Reinhartz-Berger and D. Dori, OPM/Web vs. UML – An Experimental Comparison, *Empirical Software Engineering journal* 10(1) (2005), 57-80.
50. Shan, L. and Zhu, H., CAMLE: A Caste-Centric Agent-Oriented Modelling Language and Environment, in *Software Engineering for Multi-Agent Systems III*, Ricardo Choren, Alessandro Garcia, Carlos Lucena, and Alexander Romanovsky (eds.), Springer, 2005, pp144-161.
51. E. Self and S. A. DeLoach, Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology, in: the Special Track on Agents, Interactions, Mobility, and Systems (AIMS) at The Eighteenth ACM Symposium on Applied Computing, 2003.
52. O. Shehory and A. Sturm, Evaluation of Modeling Techniques for Agent-Based Systems, in *Proceeding of the Fifth International Conference on Autonomous Agents*, 2001, 624-631.
53. K. Siau and Q. Cao, Unified Modeling Language: A Complexity Analysis, *Journal of Database Management* 12 (1) (2001), 26-34.
54. K.Siau and Y. Wang, Cognitive evaluation of information modeling methods, *Information and Software Technology* 49 (5), 2007, 455-474.
55. P. Szolovits, J. Doyle, W. J. Long, I. Kohane, and S. G. Pauker, *Guardian Angel: Patient-Centered Health Information Systems*. Technical Report TR-604. MIT Lab for Computer Science.,1994.
56. A. Sturm, *Developing and Evaluating an Object-Process Methodology-Based Multi-Agent Systems Framework*, Ph.D dissertation, Technion – Israel Institute of Technology, 2004.
57. A. Sturm, D. Dori, and O. Shehory, Specifying Communication Aspects in Multi-Agent Systems using OPM/MAS, in: the Second European Workshop on Multi-Agent Systems, Barcelona, Spain, 2004.
58. A. Sturm and O. Shehory, A Framework for Evaluating Agent-Oriented Methodologies, in: *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information System (AOIS)*, Lecture Notes in Artificial Intelligence 3030, Springer Verlag, 2003, 95-110.
59. A. Sturm, D. Dori, and O. Shehory, Single-Model Method for Specifying Multi-Agent Systems, in: *Proceedings of the Second International Joint Conference on Autonomous agents and Multiagent Systems*, 2003, 121-128.
60. A. Sturm and O. Shehory, A Comparative Evaluation of Agent-Oriented Methodologies, in F. Bergenti, M-P. Gleizes and F. Zambonelli (eds), *Methodologies and Software Engineering for Agent Systems*, Springer, 2004, 127-149.
61. A. Sturm, M. Taib-Maimon, and D. Goren-Bar, A Quantitative-Based Comparison of MaSE and OPM/MAS Design Results, *International Journal of Software Engineering and Knowledge Engineering*, Vol. 18, No.7, 2008, 933-963.
62. A. Sturm, D. Dori, and O. Shehory, Engineering Mobile Agents, in: *Proceedings of the Tenth Internatinal Conference on Enterprize Information Systems, Software Agents and Internet Computing*, 2008, 79-84.

63. J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf, Evaluation of Agent-Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform, in: Proceeding of the Fifth International Workshop on Agent-Oriented Software Engineering, 2004, 126-141.
64. Tilab, JADE - Java Agent Development Framework, <http://jade.tilab.com/>, 2008.
65. M. Winikoff and L. Padgham, Developing Intelligent Agent Systems: A Practical Guide, John Wiley and Sons, 2004.
66. M. Winikoff, L. Padgham, and J. Harland J., Simplifying the Development of Intelligent Agents, in: Proceedings of the 14th Australian Joint Conference on Artificial Intelligence, 2001 557-568.
67. F. Zambonelli, N. R. Jennings, and M. Wooldridge, Developing multiagent systems: the Gaia Methodology, ACM Transactions on Software Engineering and Methodology 12 (3) (2003), 317-370.
68. F. Zambonelli and A. Omicini, Challenges and Research Directions in Agent-Oriented Software Engineering. Autonomous Agents and Multi-Agent Systems 9, 3 (2004), 253-283.