1-1-2011

Development of a peer-to-peer information sharing system using ontologies

Ghassan Beydoun

University of Wollongong, beydoun@uow.edu.au

Graham Low

University of New South Wales, g.low@unsw.edu.au

Numi Tran

University of New South Wales

Paul Bogg

University of New South Wales

Follow this and additional works at: <https://ro.uow.edu.au/infopapers>

Part of the [Physical Sciences and Mathematics Commons](#)

Recommended Citation

Beydoun, Ghassan; Low, Graham; Tran, Numi; and Bogg, Paul: Development of a peer-to-peer information sharing system using ontologies 2011, 9352-9364.
<https://ro.uow.edu.au/infopapers/2057>

Development of a peer-to-peer information sharing system using ontologies

Abstract

Most existing agent-oriented methodologies ignore system extensibility, interoperability and reusability issues. Ontologies have been found to play a significant role in facilitating interoperability, reusability, MAS development activities (including MAS analysis and agent knowledge modeling) and MAS run-time operation (including agent communication and reasoning). However, most of the existing AOSE methodologies do not provide support for ontology-based MAS development. We present software engineering requirements for ontology-based development for MAS and examine an existing methodology, MOBMAS - a "Methodology for Ontology-Based MASs", which makes use of ontologies as a modeling tool. In this examination, we highlight how MOBMAS can be extended to utilizing ontology roles in facilitating interoperability, reusability and verification. This will be based on using MOBMAS to develop peer-to-peer (P2P) community-based information sharing application which lead us to identify ontology related steps that can be added to enhance MOBMAS. MOBMAS is used by two experienced software developers who were not authors of the methodology, to guide the development of the P2P application.

Keywords

ontologies, system, sharing, peer, information, development

Disciplines

Physical Sciences and Mathematics

Publication Details

Beydoun, G., Low, G., Tran, N. & Bogg, P. (2011). Development of a peer-to-peer information sharing system using ontologies. *Expert Systems with Applications*, 38 (8), 9352-9364.

Development of a Peer-to-Peer Information Sharing System Using Ontologies

Ghassan Beydoun¹, Graham Low², Numi Tran², Paul Bogg²

¹ University of Wollongong, Australia, beydoun@uow.edu.au

² University of New South Wales, Australia, {numi, G.Low}@unsw.edu.au

Abstract. Most existing agent-oriented methodologies ignore system extensibility, interoperability and reusability issues. Ontologies have been found to play a significant role in facilitating interoperability, reusability, MAS development activities (including MAS analysis and agent knowledge modeling) and MAS run-time operation (including agent communication and reasoning). However, most of the existing AOSE methodologies do not provide support for ontology-based MAS development. We present software engineering requirements for ontology-based development for MAS and examine an existing methodology, MOBMAS – a “Methodology for Ontology-Based MASs”, which makes use of ontologies as a modeling tool. In this examination, we highlight how MOBMAS can be extended to utilising ontology roles in facilitating interoperability, reusability and verification. This will be based on using MOBMAS to develop peer-to-peer (P2P) community-based information sharing application which lead us to identify ontology related steps that can be added to enhance MOBMAS. MOBMAS is used by two experienced software developers who were not authors of the methodology, to guide the development of the P2P application.

Keywords: Multi-agent system, methodology, ontology, problem solving methods, validation

1 Introduction and related work

A multi-agent system (MAS) consists of a group of agents, interacting with one another to achieve their goals (Wooldridge, 2002). One of the most important factors fostering MAS development is the increasing popularity of the Internet, which provides the foundation for an open environment where agents interact with each other to reach their individual or shared goals.

The demand for agent-oriented software has motivated the creation of new development approaches, such as Gaia [1], INGENIAS [2], Tropos [3] and MOBMAS [4]. A number of proposals have been made that consolidate various aspects of development approaches in an attempt to move towards a general agent-oriented approach applicable to most situations [5-7]. None adequately address

support for the issues of extensibility, interoperability and reuse other than [8] where it has been argued that an ontology-based approach is needed for a truly domain-independent agent-oriented development. Towards this, Beydoun et al. [8] proposed development of agents with domain-dependent ontologies designed with integration of problem-solving methods (PSMs) with ontology reuse.

Ontologies are an explicit formal specification of a shared conceptualization [9]. PSMs are high-level structures that describe a reasoning process employed to solve general problems [10]. A library of modular, reusable PSMs would assist the domain-independent development of agent-oriented systems as they combine with ontologies which formally describe the way a domain is perceived and are usually expressed as a collection of terms and a set of relations between the terms (Figure 1). Ontologies have been successfully used to enhance extensibility, reusability, interoperability and verify various products of software development e.g. [11-13]. Few prominent agent-oriented methodologies use ontologies in the design process [14]. When they do, their use tends to be confined to the early phase of the development (the *analysis* phase). For example, GRAMO [15] specifies how a domain model that includes goal and role analyses is developed from an initial ontology. Another example, MASE [16] uses ontologies to mediate the transition between the goal and the task analyses.

Figure 1: As new problems arise, the PSM and the ontology banks are used to construct suitable knowledge bases systems (KBS).

Towards enhancing reusability and interoperability of MAS components, our new framework [17] supports the creation of methodologies supporting and making use of ontologies throughout much of the development lifecycle. These ideas are illustrated in MOBMAS [4] – a “*Methodology for Ontology-Based MASs*” that explicitly and extensively investigates the diverse roles of ontology in MAS development and provides support for these roles. It has an ontology-based MAS development process and ontology-based MAS model definitions. MOBMAS provides support for the following key areas of MAS development: analysis, agent internal design, agent interaction design, MAS organizational design and architecture

design. MOBMAS takes advantage of existing agent-oriented methodologies, by reusing and enhancing their techniques and modeling definitions where appropriate. It endeavors to combine the strengths of the existing methodologies into one methodological framework [4, 18]. This paper is part of ongoing validation of MOBMAS. Specifically, the test case shown in this paper examines the effectiveness of MOBMAS as a tool to guide developers in producing an analysis and design of an MAS based on a given set of requirements. In particular, it evaluates how adequately does MOBMAS fulfill conceptual requirements which we identified in [8, 19]. These software engineering requirements use PSMs to map out resulting in a methodological model (Figure 2). How adequate is the realization of this model in MOBMAS will be discovered in this evaluation.

Figure 2: Ontology-based MAS development using PSMs: (1,2) Domain Ontology produces Goal Analysis which can be used to select PSMs from a PSM bank. (3, 4) Knowledge analysis can then be used to delineate local agent knowledge. (5, 6) Verification against a domain ontology and the formulation of the communication ontology (language).

The rest of this paper is organized as follows: Section 2 gives an overview of MOBMAS based on [4]. Section 3 describes the application of the Peer-to-Peer (P2P) community-based searching application and illustrates how two developers applied MOBMAS to produce an analysis of a multi agent system to implement community based searching. The developers' efforts are discussed and contrasted throughout section 3. At the conclusion of each step, we describe how to extend MOBMAS to be aligned with ontology-based development as described in this section. Section 4 provides further analysis of these efforts. Section 5 concludes with a description of further extensions to this research.

2. MOBMAS Methodology

In MOBMAS [4], the MAS development starts with a domain ontology which is initially used to identify goals and roles of the system to index an appropriate set of problem solving capabilities from an appropriate existing library of capabilities. Individual ontologies corresponding to the knowledge requirements of each capability are then extracted from the initial common ontology, to provide knowledge representation and allow reasoning by individual agents. Those ontologies form the basis for an iterative process to develop a common communication ontology between all agents and verify the knowledge requirements of chosen capabilities. Individual localised ontologies may also require incremental refinement during the iterative process. Appropriate ontology mappings are needed between local ontologies and the communication ontology. The development of MAS using MOBMAS has five activities (Figure 3). Each focuses on one of the following key area of MAS development: Analysis, Organization Design, Agent Internal Design, Agent Interaction Design and Architecture Design. The development process of MOBMAS is highly iterative.

MOBMAS activities are detailed as follows:

Analysis Activity: This activity aims to form a conception of the target MAS from the domain ontology and the system requirements, giving a first-cut identification of the roles and tasks that compose the MAS. This activity consists of developing the following five models: System Task Model, Organizational Context Model, Role model, Ontology Model and as well as identification of Ontology-Management Role. The Role Model is developed in a highly iterative manner with the System Task Model, given the association between roles, role tasks and system tasks. The Ontology Model is used to refine and validate those models (and vice versa). This activity also specifies the ontological mappings between the MAS Application Ontologies.

MAS Organization Design: This activity refines the organizational structure of the target MAS and identifies a set of agent classes composing the system. If the MAS is a heterogeneous system that contains non-agent resources, these are also identified and their applications are conceptualized. This activity consists of the following four steps: Specify the MAS Organizational Structure, Develop the Agent Class Model; Develop the Resource Model; and Refine the Ontology Model of the previous activity. The developer also specifies the mappings between Resource Application Ontologies and relevant MAS Application Ontologies, to enable the integration of these resources into the MAS application and to support the interoperability between heterogeneous resources.

Agent Internal Design: For each agent class, this activity specifies belief conceptualization, agent goals, events, plan templates and reactive rules. It consists of the following five steps: Specify Agent Class' Belief Conceptualization identifying which part(s) of the Ontology Model are needed by an agent class to conceptualize its run-time beliefs; Specify Agent Goals identifying the states of the world that an

agent class aims to achieve or satisfy using the Role Model; Specify Events in the environment that agents need to respond to at run-time; Develop Agent Behaviour Model specifying how each agent class behaves to achieve or satisfy each agent goal as planning behaviour or reactive behaviour; and Update the Agent Class Diagram with the details identified in the previous three steps. The Agent Behaviour Model is checked for consistency against the Ontology Model and vice versa.

Figure 3. MOBMAS development process: The solid arrows represent the flow of steps within and across activities, while the dotted arrows indicate the potential iterative cycles of steps. Models produced or refined by each step are shown in the square brackets.

Agent Interaction Design: This activity models the interactions between agent instances, by selecting a suitable interaction mechanism for the target MAS and modeling the interactions. It has two steps: Decide which interaction mechanism is best suited to the target MAS (direct or indirect); and then Define how agents interact depending on the selected interaction mechanism. The resultant Agent Interaction Model is represented by a set of Interaction Protocol Diagrams. The developer validates the Agent Interaction Model against the Ontology Model. The Agent Class Model is also checked to ensure that all communicating agent classes share the same application ontologies that govern their interactions. Lastly, the Agent Relationship Diagram is updated to show descriptive information about each interaction pathway between agent classes.

Architecture Design: This activity deals with various design issues relating to agent architecture and MAS architecture. It has the following five steps: Identify Agent-Environment interface requirements; Select Agent Architecture for the most appropriate architecture(s) for agents in the MAS; Specify MAS Infrastructure facilities identifying system components that are needed to provide system-specific services; Instantiate agent classes; and Develop MAS deployment plan.

MOBMAAS will be further illustrated in the rest of this paper in applying it to develop a community based search engine. In addition, we identify extensions to MOBMAAS to accommodate the ontology based paradigm combined with the appropriate usage of PSM (as described in Section 1).

3 Community-based P2P Information Sharing MAS

In this section, we illustrate the use of MOBMAAS on a P2P information sharing application by two experienced system developers. An implicit semantic web developing at the heart of the system guides the interactions as in [20, 21].

3.1 Application Description

Each human user of the system is represented by an agent in the network to act on his/her behalf. This agent locates files and responds to queries by other similar agents. The collection of all these agents and agents assisting them in their tasks form the P2P community based searching MAS (Figure 4). An agent representing the human user has access to a knowledge base containing electronic files that the user is willing to share with other users. Each file is identified by its title and type (e.g. HTML, pdf, music or video).

As agents interact on behalf of their users, communities of interest begin to emerge. These communities may overlap (Figure 5). A human user may belong to

more than one community, for instance an IS researcher may also be a cinema goer. Agents develop an awareness of the communities to which users belong and use this awareness to fulfil their users search requests efficiently and effectively, by interacting with the agents in the communities most likely to be able to serve their requests.

Figure 4. The P2P Multi agent system is the collection of the agent assistants and any supporting specialized agents

Figure 5. A human user may belong to more than one community.

A human user can pose a query to request files. Each query is made up of one or more keywords. The P2P system is responsible for locating sites where files matching the queries may reside, based on the behavior of the users at those sites (as represented by their agents). The mediation between the human users is always done by the system and is initiated by the agent representing the human making the

request. The agent of the like-minded user responds either by providing details about the files it can supply, or refusing the service. When all responses are received, the agent combines and refines the results to compose a list of files that satisfy the query. The agent initiating the query can then select which file(s) it wants to download to the human it represents and initiates the file transfer process. After a successful transfer the knowledge base, located where the query was made, is updated to contain the received file(s).

For all agents involved in processing the query, their knowledge base is also updated with additional information reflecting the interests of the agent which initiated the query. This information is used in future queries. That is, as agents interact they develop awareness of the files possessed by their peers and which peers may be interested in the files that they themselves have.

At each node in the network, each user-agent keeps a record of its history of information sharing. The history contains two records: one of the past queries that it made on behalf of the human user and its respective responders, and one of the past queries received and their respective agent senders (acting on behalf of other human users). The former needs to be updated every time the user-agent receives a result list from the system, while the latter requires update every time the user-agent replies to a query sent by the system. The history is used to produce short lists of candidate nodes for future queries, by calculating the similarity between the current query and a past query [21]. If no nodes can be short-listed, or if all candidate user-agents do not provide the service required, the agent-user broadcasts the query to a wider circle of user-agents in the community, to identify new candidate providers. In a fully evolved P2P system, agents may use their knowledge about other users interests to request/negotiate for information from their peers when they do not know who has the files of interest. Any new providers are eventually added to the history, thereby expanding the user-agent's contact circle. This strategy of information sharing can be applied to any domain. We limit our analysis to an application for the Movies domain. This simplifies the requirements of the system by focusing on one community, and details of how a community emerges or connect to another community (using a global ontology) is left out for future extension. Accordingly, the information to be shared amongst user-agents is assumed to only be movie-related files, such as movie trailers, movie posters or movie web pages.

3.2 P2P Analysis

The Analysis activity first step is to construct a System Task Model to specify the required system functionality and its decomposition structure. For the P2P information sharing application, Developer #1 specifies the core system task as "Satisfy file-sharing request", which is composed of two sub-tasks "Process user search query" and "Carry out file-transfer process". Each of these sub-tasks is further decomposed into smaller-grained sub-tasks.

Developer #2 produces the system task model shown in Figure 6 (b). To contrast both task models: Developer #1 includes the maintenance of the sharing history in his task model whilst Developer #2 does not. The latter also uses two perspectives to identify the tasks for a given agent, the upload (server) and the download (client) perspective. The next step in the Analysis activity of MOBMAS is to investigate the organizational context in which the MAS will reside and support, to elicit any existing organizational structure that the MAS may imitate. In this application, this step is omitted by both developers as the agents of the target MAS do not represent any humans within a human organization.

Figure 6. System Task Diagram

The next step is to construct a Role Model. Developers identify roles by grouping closely related tasks in the System Task Model. Developer #1 assigns all tasks dealing with user interactions to a “User Interface” (Figure 7a) and all tasks related to the file-transfer history to a “History Manager” role. He also defines an “Information Retriever” to handle all tasks related to processing user query and file transfer and a “Portal” role was identified to act as a broker in the P2P network, by identifying new information providers when required. Developer #1 also recognises the need for *Wrapper* roles to enable operating in a heterogeneous environment.

Figure 7. Role Diagram .

Developer #2 assigns all the tasks dealing with interactions between the human user and his representing agents to a “User Interface Manager” role (Figure 7b). The tasks relating to user query processing and file transfer (including file downloading and uploading) are allocated to three separate roles, “Searcher”, “Downloader” and “FileServer”. The tasks relating to maintaining transfer histories are implicitly handled by the roles “Downloader” and “FileServer”.

Figure 8. Ontologies used: Movie Ontology (based on DAML ontology at <http://www.cse.dmu.ac.uk/>) and information retrieval ontology.

A key difference between the two role models is that Developer #1 recognises early in his analysis that the system will operate in a heterogeneous environment and includes the role “Wrapper” to mediate between agents and non-agents. Devel-

oper #2 as we will see in the next section, adds this role during the next activity of MOBMAS.

An Ontology Model is then constructed to define the necessary application ontologies for the target MAS. At this stage, only MAS Application ontologies are examined. Resource Application ontologies are identified later in the Organization Design activity (shown in the next section). An information sharing MAS for the Movies domain would require two MAS Application ontologies: one for conceptualizing the Movies domain and one for conceptualizing the domain (Figure 8). All agents in the system are expected to know and use these two Application ontologies which are not expected to change at run-time, hence the developers decided that the ontologies should be stored at some publicly-accessed ontology servers and be accessed freely by all agents. Both developers recognise that no particular role or agent is therefore needed to manage or control these servers and neither adds any further roles to their corresponding models. For the completion of ontology-based development, these ontologies will be the handles that index appropriate PSM from appropriate repositories (as described in Figures 1 and 2).

3.3 P2P MAS Organization Design

The first step in this activity is to refine the Role Model developed in Section 3.2 to specify authority relationships between roles. Developer# 1 identifies the following relationships (shown in Figure 9 a): role “Information Retriever” has a peer-to-peer relationship with all roles except with the role “History Manager” (expressed by a “control” association in Figure 9a) because role “Information Retriever” has the authority to delegate work to role “History Manager”, and the latter is obliged to perform the delegated tasks and should not reject a request from the former.

The second step is identification of agent classes. Developer #1 tentatively identifies Agent classes from roles via one-to-one mappings producing five different agent classes (User Interface, Information Retriever, History Manager, Mediator and Wrapper) in the P2P MAS with each of the classes assuming a single role. A preliminary Agent Relationship Diagram is constructed to show the tentative classes with their roles and acquaintances (i.e. interaction pathways) (Figure 7a). These acquaintances are derived directly from the acquaintances between roles assumed by the identified classes (cf Figure 9a).

For the first step, Developer #2 associates two roles “Searcher” and “Downloader” (Figure 9b) into one single agent class “Client” (Figure 10). He does not use the portal notion as Developer #1 does to generalize the interface role to interfacing to communities of agents. Developer #2 focuses on client server roles by introducing a role *client* that can switch between the two perspectives. For the second step, Developer #2 constructs a preliminary Agent Relationship Diagram to show the tentative agent classes, their roles and their interaction pathways (Figure 7b). Note that the explicit separation of the “Client” and “Server” agent classes in the

solution helped him to clearly model that each user in the P2P network can be both a client and a server. Each user in this model can be represented by a “Client” agent at one time, and by a “Server” agent some other times.

The next step of the organisation activity is to identify the non-agent software resources in the analysis phase. The following resources are identified by both developers: knowledge sources containing movie-related files, e.g. web servers of HTTP files.. Each knowledge source is to be managed and controlled by a specialized wrapper agent which provides an interface to the resource when requested by other agents in the system. Accordingly, the ontology conceptualizing each knowledge source is defined and thereafter added to the Ontology Model. This wrapper agent is already identified by Developer #1 in the analysis activity whilst Developer #2 updates his Agent Class Model to show it as a newly identified class.

Figure 9. Agent Relationship Diagram (Note: diagram progressively updated throughout design – final diagram shown to reduce space)

Following the *MAS Organizational structure*, assigning Problem Solving Methods to tasks identified from roles within the organizational structure is required. This will be used in the development of the agent class model – the consideration of methods to solve a group of tasks may assist in identifying the agent classes. A suggested technique for *PSM assignment* in [22] is:

1. For each task, search the PSM library to find a suitable PSM definition and *assign* it to the task. If no suitable PSM definition can be found, a new method may need to be designed at the *Agent Internal Design* phase.
2. For each PSM definition assigned to a task, assess existing dependencies with other PSM definitions. A dependent PSM definition may suggest the existence of tasks that were not identified.
3. Iteratively revise the assignment of PSM definitions to tasks (if necessary) to ensure consistency between PSM dependencies.

As a consequence of the assignment of PSMs to tasks, missing tasks (or possibly missing requirements) may be identified. This may suggest a revision of the task model in the *Analysis* phase.

3.4 P2P Agent Internal Design

The internal design of each agent class starts with the identification of the agent class' belief conceptualization; that is, the identification of ontologies conceptualizing the agent's potential run-time beliefs. For example, the "Information Retriever" agent class in Developer #1's design requires to commit to two ontologies: "Movie Ontology" and "Information Sharing Ontology", because at runtime agents of this class require knowledge about both the movie domain and the information sharing domain. In addition to those two ontologies, the "Wrapper" agent class should commit to the "MovieTrailer Resource Ontology" because wrapper agents need as well access the MovieTrailer knowledge source at run-time. Agent goals are identified directly from role tasks. However, while role tasks are specified using imperatives, agent goals are specified in the form of "*something is achieved*". For instance, role "History Manager" in Figure 7 has a task "Identify candidate providers from file-sharing history". This task indicates a goal "*Candidate providers are identified from file-sharing history*" for agent class "History Manager".

Both developers identify events affecting agents' courses of actions. For example, Developer #1 identifies "Reception of user query from User Interface agents" (which activates an agent goal "Answer is found for user query"), "Input of user's file selection" (which activates an agent goal "File is downloaded") and "Input of user's cancellation" (which signals the agent to forfeit its active goal) as events that need to be taken into account by "Information Retriever" agents. Both developers

extend the Agent Class Model to show the listing of belief conceptualization, agent goals and events for each individual agent class (Figure 10a and 10b).

The developers next produce an Agent Behaviour Model to define agent plan templates and reflexive rules for each agent class to achieve its agent goals. Both planning and reactive behaviour are considered for each agent class, in respect of each agent goal. Following are examples: the “History Manager” agent class of Developer #1 requires planning behaviour to fulfill the corresponding agent goal “*Candidate providers are identified from file-sharing history*”; the “User Interface” agent class of Developer #2 employs reactive behaviour to achieve the goal “User query is accepted”.

Figure 10. (a) Class Diagram for “Information Retriever” agent class (Developer #1). (b) Agent Class Diagram for “Server” agent class (Developer #2)

Using MOBMA, the Agent Behaviour Model is validated during design and analysis against the Ontology Model, to ensure that the datatypes of all variables in the agent plan templates and reflexive rules are equivalent to the ontological concepts defined in the “Movie Ontology”, “Information Sharing Ontology” and “MovieTrailer Resource Ontology”. For example, variable “q” in has a datatype “UserQuery”, which is a concept in the “Information Sharing Ontology” (Figure 8). The Agent Class Diagram is also checked to ensure that the agent class’ belief conceptualization contained the ontology involved (in this case, the “User Interface” agent class should specify the “Information Sharing Ontology” in its belief conceptualization).

An Agent Behaviour Model is developed to define plan templates and reactive rules for each agent class. Each of the developers produces plans modelling the behaviour of their agents. Examples are shown in Figures 11 and 12: Figure 11 shows

an elaborate plan for history management by Developer #1 (to be used by History Manager agent), Figure 12 shows a less complex plan for uploading request by Developer #2 (to be used by the Server agents). Plans are also illustrated by plan diagrams that are used by both developers. These were used for simpler plans (Figure 13 and 14). For example Developer #1 did not produce a plan diagram for the plan shown in Figure 11. The developers consider both planning and reactive behaviour for each agent class, depending on the target agent goal.

<p>Initial state: <i>file history', IR_Message1 is received</i></p> <p>Target agent-goal: <i>file history'', IR_Message2 is sent.</i></p> <p>Commitment strategy: <i>single-minded</i></p> <p>List of sub-agent-goals (if any): <i>OntologyConcept_Located, ClosestFileLocated, HistoryUpdated</i></p> <p>List of actions (if any):</p> <p style="padding-left: 40px;">If RetrieveFile (MatchKeyword (keyword, history)) == NULL</p> <p style="padding-left: 80px;">IR_Message2 = (MatchKeyword (keyword, history), NULL)</p> <p style="padding-left: 40px;">Otherwise</p> <p style="padding-left: 40px;">IR_Message2 = (keyword, Retrieve (MatchKeyword (keyword, history)))</p> <p style="padding-left: 40px;">Send IR_Message2</p> <p style="padding-left: 40px;">MatchKeyword (keyword, history): ontology_concept</p> <p>Pre-condition: <i>keyword is an informal concept of the ontology</i></p> <p>Post-condition: <i>ontology_concept is the closest match to keyword in the ontology</i></p> <p style="padding-left: 40px;">RetrieveFile (ontology_concept, history): File_pointer</p> <p>Pre-condition: <i>ontology_concept is a formal concept of the ontology</i></p> <p>Post-condition: <i>File_Pointer points to most updated version of a file relevant to the concept, otherwise it is NULL.</i></p> <p style="padding-left: 40px;">UpdateNode (ontology_concept, agent_ID)</p> <p>Pre-condition: <i>ontology_concept is a concept within the ontology</i></p> <p>Post-condition: <i>agent_ID is included in the history as an agent who enquired about the ontology_concept</i></p> <p>Events: <i>IR_Message1 (Keyword_query, file_pointer?, agent_ID), IR_Message2 (keyword_response, fptr)</i></p> <p>Conflict resolution strategy (if applicable): <i>NA</i></p>

Figure 11. Agent Plan template for history manager (Developer #1): First receive (*Keyword_query, file_pointer?*) requesting the closest file pointer matching a *keyword*, *keyword_query*. Secondly, the history manager agent browses the history, if successful it returns the tuple (*keyword_query, file_pointer*) otherwise it looks for the closest, then the

closest matched keyword, keyword_response and returns (keyword_response, file_pointer), in this second scenario file_pointer may be NUL .

Initial State: any **Agent Goal:** File details provided upon receiving an upload query
Commitment Strategy: single-minded
Action 1: ValidateQuerySyntax (q : UserQuery)
 Pre-condition: true *Post-condition:* Query q is valid OR refusal message is sent to cl: Client
Action 2: ExecuteQuery (q : UserQuery)
 Pre-condition: q is valid *Post-condition:* File queried in q is found or no result is found
Action 3: ReplyToQuery (filepointer: File)
 Pre-condition: true *Post-condition:* f.Filename and f.Filetype are sent to cl: Client
Event 1: Reception of upload query q

Figure 12. Agent Plan Template for “Server” agent class (Developer #2)

Figure 13. Reflexive Rule Specification for “User Interface” agent class (Developer #1)

Figure. 14. Agent Plan Diagram (for Server agent class) (Developer #2)

The *Specify agent goals* step has been defined in MOBMAS to translate tasks into agent goals. The translation of tasks into goals may influence the selection of PSM definitions assigned to each task. The developer is recommended to check that PSM definitions are consistent with the formulation of agent goals.

To complete the ontology-based development with PSMs, a *PSM orchestration model* is required so that for each agent, PSM definitions are refined to each agent goal for the agent's domain ontology. Techniques have been recently suggested in [22] which produce a set of *PSM mappings* for the agent that adequately cover every task/goal required by the agent to be performed. The PSM mappings may suggest that the agent class model needs revision. In this case, the PSM mappings may suggest the iterative refinement of the agent class model in the *MAS Organizational Design* phase.

The *Agent behaviour model* step is the last to be performed. PSM mappings are highly domain-specific, task-oriented knowledge structures, and plans are not necessary where task/goals are assigned a PSM mapping. However, for task/goals without assigned PSM mappings, custom methods need to be designed. The design follows the MOBMAS prescription for determining whether plan templates or reflexive rules are selected.

3.5 P2P Agent Interaction Design

The first step in this activity is to select a suitable interaction mechanism for the P2P MAS. The "direct" interaction mechanism (using ACLs) is chosen by both developers over the "indirect" mechanism (using tuplespace/tuple-centre), because the speech-act performatives provided by ACLs are expected to support a higher level of communication semantics than the level provided by the Linda-like primitives used by the tuple-space mechanism.

Each of the developers develops an interaction model for every plan he has. For the example plans shown in the previous section, the corresponding interaction model is shown later in this section. This so-called Agent Interaction Model (in MOBMAS) is developed to define interaction protocols between agent instances. Each protocol is depicted by an AUML Interaction Protocol Diagram (Figure 15 for Developer #1 and Figure 16 for Developer #2).

Developer #1 expresses his interaction protocol (corresponding to Figure 15) in the following format as well:

```

    UI sends (Keyword) to IR
    IR sends (keyword) to HM
    HM sends (keyword2, ?, agent_ID) /* keyword2 is the closest available that
    occurred in the past, where agent_ID made a request using it */
    IR sends (keyword) to PA /* PA is given by agent_ID */
    /* PA broadcasts */
    /* PA chooses agent_ID2 and sends it to IR */

```

PA sends (keyword, agent_ID2) to IR
IR sends (keyword, agent_ID2) to agent_ID2 / agent_ID2 is a peer of IR */*
IR receives (keyword, file_ptr, agent_ID2)
IR sends (keyword, file_ptr) to UI / download starts */*
IR sends (keyword, file_ptr, agent_ID2) to HM / history update */*
Where each of the communicative events follow the following format: (Time Stamp, Sender Agent, Message, Receiver Agent, Comment)

Figure 15. Interaction Protocol (for Developer #1) for updating history manager

Figure 16. Interaction Protocol Diagram (from developer 2 for getting file from candidate provider)

The developers subsequently check their Agent Interaction Model against the Ontology Model, to make sure that the datatypes of all variables in the interaction protocols are equivalent to the ontological concepts in the “Movie Ontology” or “Information Sharing Ontology” (except for basic datatypes like string and integer). After constructing the Agent Interaction Model, the Agent Relationship Diagram is updated to show various descriptive information about each interaction pathway between agent classes (Figure 10), as follows:

- the name of the Interaction Protocol Diagram depicting the protocol governing the interactions; and
- the name of the ontology used to define the interactions’ semantics.

To complete the ontology-based development advocated in the model shown in Figure 2, *PSM interaction refinement* needs to be performed after the selection of the interaction mechanism. The interaction mechanism specifies what communication protocol is to be used. Interaction refinement uses the communication protocol and domain ontologies to identify and refine Interaction Protocols, Model, and Strategy PSMs. Suitable techniques have recently been suggested in [22]. The work products of this step would be a set of *protocol mappings*, *model mappings* and *strategy mappings* consistent with the work products from the *Agent Internal Design* phase. The protocol mappings would provide structure to agent-to-agent message passing in the language of the communication protocol. This includes what type of parameters can be passed, and the permissible sequences of message

exchanges. A strategy mapping for a particular interaction protocol defines a selection of possible messages that an agent might send at a particular point in time. In other words, strategy models provide options for message exchange, and interaction protocol provides message exchange restrictions. The interaction model is defined in terms of the protocol mappings and strategy mappings for each agent.

3.6 Architecture Design

The MOBMAS guidelines in support of architecture are followed: The first step is the identification of the requirements of agent perception, effect and communication mechanisms. For our example application, the environment does not contain any physical objects or hardware interfaces with the agents, this reduces this step to connecting the “User Interface” agents to an elaborate user-interface component, as this is the only means of getting inputs and providing outputs to human users in this application. The “Wrapper” agents should also be equipped with an ability to connect to its wrapped knowledge sources. With regard to communication, the implementation platform should be able to support the exchange of rich multimedia files. An Agent-Environment Requirement Specification is constructed to document these characteristics and requirements.

The next step is to determine the appropriate agent architecture to the target MAS. Since agent classes in the system use both planning behaviour and reflexive behaviour for achieving their goals, a hybrid architecture is clearly required. Various hybrid architectural solutions are available off the shelf for use e.g. [23, 24, 25].

The target MAS would require basic network facilities such as agent naming service, agent creation/deletion service and security service. Common coordination facilities such as agent directory service and message transport service are also required. Necessary knowledge facilities were ontology servers, protocol servers and problem-solving methods servers. The instantiation cardinality of each agent class is also determined. All agent classes in the P2P application were instantiated with a “+” cardinality; that is, each class has one or more agent instances at implementation time. The Agent Class diagrams are updated to show this instantiation configuration.

In terms of a ontology-based AOSE methodology, a runtime capability library may exist whereby Problem Solving Method mappings may be used to directly infer runtime capabilities available within the library. The domain ontology can be used in the identification and selection of runtime capabilities for agent class instantiation. Details of this is not discussed here, and left as future work.

4 Discussion and Summary

The underlying conceptual models that each of the two developers used are clearly different. Developer #1 analysis showed a peer-to-peer orientation as compared with Developer #2 who seemed to have a client-server model ghost haunting in the background. This led to clear differences in their analyses model and later during their design. This was evident in contrasting their analyses and in particular its impact on the task and role models. For instance, the system task model for Developer #1 includes the maintenance of the sharing history in his analysis whilst Developer #2 does not. Developer #2 also uses two perspectives to identify the tasks for a given agent, the upload (server) and the download (client) perspective. In another instance, Developer #2 associated two roles “Searcher” and “Downloader” into one single agent class “Client” whilst Developer #1 introduced the role portal which generalizes the interface role to interfacing to communities of agents. This difference in the underlying model was also evident in the complexity of the indicative plan that each of the developers provided in the internal models of their agents: Developer #1 provided a fairly complex plan to update the history of interactions whilst Developer #2 provided a much simpler plan.

Withstanding the subtle differences in the models provided, the work of the developers clearly demonstrated MOBMAS – a methodology for ontology-based MAS development – on a P2P community-based information sharing application. In addition, it illuminated the way to identify required extensions to accommodate the conceptual steps described in section 1 and 2 of the paper. This demonstration is the more significant in that the developers were not actually authors of the MOBMAS methodology but were given detailed documentation of the methodology. Their responses to a detailed questionnaire regarding the usage of MOBMAS, indicated that they valued the step-by-step development process of MOBMAS and the provision of many heuristics and techniques to support each step. MOBMAS is an iterative (spiral) methodology which guides developers to revisit their models to complete any details they overlooked. An example of how useful this feature of MOBMAS is evident in the iteration between analysis and MAS organisation. For example, Developer #2 overlooked the role of the *wrapper* in his analysis but revisited this and added this role during the MAS organisation activity. MOBMAS also provided *verification and validation*: The steps of MOBMAS enforce consistency checking amongst the major model kinds. For example, the ontology Model is used to verify and validate the System Task Model, Agent Class Model, Agent Behaviour Model and Agent Interaction Model. Currently, we do not have a tool to enforce this checking. We are in the process of formalizing the current manual checking. This will be usable as a stepping stone to develop a tool.

To summarise, this paper operationalised a theoretical framework of ontology-based development of MAS based on reusable PSM components. It has done this by using and extending an existing methodology which uses ontologies, MOBMAS. Each of the first four phases of MOBMAS was demonstrated and ex-

tended. We applied MOBMAS on a P2P community-based information sharing application. The work confirmed the ease of use of MOBMAS and its support for validation and verification but it highlighted its limitations in guiding developers in allocating appropriate PSMs to agents as required by the domain ontology. Following each step in development of a P2P community sharing domain, the paper highlighted how each step can be used to use PSM components.

We still need to confirm that systems developed with MOBMAS are interoperable and extendible. We plan to deploy a completed P2P system in a heterogeneous environment to validate its interoperability. With respect to extensibility, we are intending to develop a webportal using MOBMAS and then vary the initial requirement of the system and assess how easy it is for developers to modify the original portal. We anticipate that because new knowledge sources and agents can be easily added to the MAS shown in this paper, and since in any application core models of MOBMAS are composed of ontologies and ontological concepts (namely, Agent Belief Conceptualization, Agent Behaviour Model and Agent Interaction Model), a design can be adapted to a new application by changing the ontologies involved. However, some further details in MOBMAS need to be worked out in order to manipulate ontologies within the development process. For example, in case two ontologies with different conceptualization of the same domain are used during the requirement changes, then ontology mappings would be required.

5. Conclusion and Future Work

MOBMAS methodology was initially reviewed and refined based on the feedback of two experts in AOSE. The MOBMAS used here is the refined methodology. The feedback from the developers was used to refine MOBMAS into its final version. Both the expert reviews and test-uses by developers were conducted in a sequential order. Evaluation of the first expert/developer was used to refine MOBMAS before the second expert/developer was asked to evaluate/use the refined version. This sequential and independent procedure prevented the possibility of two experts/developers identifying the same areas for improvement, and helped to identify new areas of improvement that might arise from the refinement of the methodology after the first review/test-use. In addition, the refinements made to MOBMAS as a result of the second expert's/developer's feedback were also discussed with the first expert/developer to ensure that no conflicts of opinions occurred. A feature analysis was conducted on the final version of MOBMAS to verify MOBMAS' ability to support important AOSE methodological features, steps and modeling concepts. A key feature of this validation, that it was driven by an ontology-based development conceptual framework and required steps were identified for ontology/PSM.

Limits on the evaluation will remain without a direct comparison with other methodologies across a number of application types. We expect this next phase of

evaluation to highlight in addition to its ease of use, its interoperability and its extensibility and it could make use of recent work on evaluation of cooperatively developed models e.g. in [27]. MOBMAS supports interoperability for agents with heterogeneous local knowledge can communicate by sharing a common MAS Application Ontology, and by using this ontology to formulate and interpret their exchanged messages. This also leads to extensibility since new knowledge sources and agents can be easily added to the MAS. MOBMAS also supports reusability since the core models of MOBMAS are composed of ontologies and ontological concepts (namely, Agent Belief Conceptualization, Agent Behavior Model and Agent Interaction Model), hence the design can be adapted to a new application by simply changing the ontologies involved. This feature of MOBMAS will be tested by attempting to reuse developed work products in significantly different applications.

Finally MOBMAS provided verification and validation: The steps of MOBMAS enforce extensive consistency checking amongst the major model kinds. For example, the Ontology Model is used to verify and validate the System Task Model, Agent Class Model, Agent Behavior Model and Agent Interaction Model. Currently, we do not have a tool to enforce this checking. We are in the process of formalizing the current manual checking. We recently in [8] completed a preliminary framework which checks the early requirements against role models. This will be usable as a stepping stone to develop a tool (as suggested by one of the reviewers). Ontology-based development steps required for PSM reuse as identified will be cemented by creating appropriate processes and workproducts. This work has already started in [22, 26].

References

1. Wooldridge, M., *et al.* (2000) The Gaia Methodology for Agent-Oriented Analysis and Design. In *Autonomous Agents and Multi-Agent Systems*, pp 285-312, Kluwer Academic Publishers.
2. Pavon, J. and J. Gomez-Sanz (2003). *Agent Oriented Software Engineering with INGENIAS*. 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003), Prague, Czech Republic.
3. Bresciani, P., *et al.* (2004) TROPOS: An Agent Oriented Software Development Methodology. *Journal of Autonomous Agents and Multi-Agent Systems* 8, 203-236
4. Tran, Q.N.N., and Low, G.C. (2008) MOBMAS: A Methodology for Ontology-Based Multi-Agent Systems Development, *Information and Software Technology Journal*, 50 (7), 697-722.
5. Fitzgerald, B., *et al.* (2003) Software Development Method Tailoring at Motorola. *Communications of the ACM* 46, 65-70
6. Glass, R.L. (2004) Matching methodology to problem domain. *Communications of ACM* 47, 19-21

7. Glass, R.L. (2000) Process diversity and a computing old wives'/husbands' tale. *IEEE Software* 17, 127-128
8. Beydoun, G., Krishna, A.K., Ghose, A. and Low, G.C., 2008, Towards ontology-based MAS methodologies: ontology based early requirements, in *The Inter-Networked World: ISD Theory, Practice, and Education* (eds. C. Barry, M. Lang, W. Wojtkowski, G. Wojtkowski, S. Wrycza and J. Zupancic), Springer-Verlag, New York.
9. Gruber, T. R. (1993) A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, 5(2), 199-220.
10. Rodríguez, A., *et al.* (2003) Experiences in Reusing Problem Solving Methods - An Application in Constraint Programming. *Proceedings of Knowledge-Based Intelligent Information and Engineering Systems, 7th International Conference, KES2003*, 1299-1306.
11. Shave, M. J. R. (1997). "Ontological Structures for Knowledge Sharing." *New Review of Information Networking* 3: 125-133.
12. Uschold, M. and M. Grueninger (1996). "Ontologies: Principles, Methods and Application." *Knowledge Engineering Review* 11(2): 93-195.
13. Fensel, D., E. Motta, S. Decker and Z. Zdrahal (1997). *Using Ontologies for Defining Tasks, Problem-Solving Methods and Their Mappings*. 10th European Knowledge Acquisition Workshop (EKAW97), Spain, Springer.
14. Tran, Q. N. and G. Low (2005). *Comparison of Methodologies*. Agent-Oriented Methodologies. B. Henderson-Sellers and P. Giorgini. Hershey, Idea Group Publishing: 341-367.
15. Girardi, R., C. G. d. Faria and L. Balby (2004). *Ontology-based Domain Modeling of Multi-Agent Systems*. OOPLSA Workshop.
16. Dileo, J., T. Jacobs and S. Deloach (2002). *Integrating Ontologies into Multi-Agent Systems Engineering*. 4th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS2002), Italy.
17. Tran, Q. N. N., Low, G. C. and Beydoun, G. (2006), A Methodological Framework for Ontology Centric Agent Oriented Software Engineering , *International Journal of Computer Science and Engineering*, 21, 117-132.
18. Tran, Q.-N. N., G. Low and M.-A. Williams (2004). *A preliminary comparative feature analysis of multi-agent systems development methodologies*. AOIS 2004.
19. Beydoun, G., Tran, N., Low, G. and Henderson-Sellers, B., 2006a, Foundations of ontology-based MAS methodologies, *Procs. AOIS2005* (eds. M. Kolp, P. Bresciani, B. Henderson-Sellers and M. Winikoff), LNAI 3529, Springer-Verlag, Berlin, 111-123.
20. Beydoun, G., Kultchitsky, R., Manasseh, G., (2007), *Evolving Semantic Web with Social Navigation*, *Expert Systems with Applications* 32: 265-276.
21. Beydoun, G., *Formal concept analysis for an e-learning semantic web*, *Expert Systems with Applications* 36 (8), 265-276.
22. P. Bogg, G. Beydoun, G. Low, Problem-Solving Methods in Agent-Oriented Software Engineering, in *Information Systems and Development Conference 2009 (ISD09)*, China.
23. I.A. Ferguson: *Turing Machines: An Architecture for Dynamic, Rational, Mobile Agents*. 1992, University of Cambridge: Cambridge.
24. R.J. Firby: *Adaptive Execution in Dynamic Domains*. 1989, Yale University: Yale.

25. E. Gat: Integrating Planning and Reacting in a Heterogenous Asynchronous Architecture for Controlling Real-World Mobile Robots, in *10th National Conference on Artificial Intelligence*. 1992. San Jose, California: The MIT Press.
26. Beydoun, G., Low, G., Henderson-Sellers B, Mouratidis, H., Gomez-Sanz, J. J., Pavon, J., Gonzalez Perez, C “FAML: a Generic Metamodel for MAS development,” *IEEE Transactions on Software Engineering*, vol. 35 (6), 2009.
27. Beydoun, G., Hoffmann, A., Simultaneous modeling and knowledge acquisition using NRDR. *PRICAI'98: Topics in Artificial Intelligence. Lecture Notes in Computer Science*. Volume 1531. 1998. Springer. 83-95.