Preliminary Validation of MOBMAS (Ontology-Centric Agent Oriented Methodology): Design of a Peer-to-Peer Information Sharing MAS

Quynh-Nhu Numi Tran[1], Ghassan Beydoun[2], Graham Low[1], and Cesar Gonzalez-Perez[3]

[1] School of Information Management and Technology Management, University of New South Wales, Australia
{g.low,numitran}@unsw.edu.au
[2] Faculty of Informatics, University of Wollongong, Australia
beydoun@uow.edu.au
[3] Faculty of Information Technology, University of Technology of Sydney, Australia
cesargon@it.uts.edu.au

Abstract. Most existing AOSE 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 modelling) 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. In light of this shortcoming of the existing AOSE work, we have developed MOBMAS – a "*M*ethodology for *O*ntology-*B*ased *MAS*s". In this paper, as part of its ongoing evaluation, we demonstrate MOBMAS on a peer-to-peer (P2P) community-based information sharing application. MOBMAS is used by an experienced software developer, who is not an author of the methodology.

Keywords: Multi-agent system, methodology, validation.

1 Introduction

Ontologies are an explicit formal specification of a shared conceptualization [1]. They have been successfully used to enhance extensibility, reusability, interoperability and verify various products of software development e.g. [2-4]. In Agent-Oriented Software Engineering (AOSE), existing methodologies do not implement these important potentials of ontologies and very few MAS methodologies include ontologies in their workproducts and processes (as shown in the recent survey of existing AOSE methodologies [5]).

When AOSE methodologies use ontologies, this use tends to be confined to the early phase of the development (the *analysis* phase). For example, GRAMO [6] specifies how a domain model that includes goal and role analyses is developed from an initial ontology. Another example, MASE [7] uses ontologies to mediate the transition between the goal and the task analyses. Our use of ontologies in developing MAS is perhaps closest to recent work in [8] which recognizes the role of using ontologies for verification of models during the analysis phase. Outside the analysis phase, ontologies currently are mainly used to express a common terminology for agent interactions in an MAS e.g. [9].

Towards enhancing reusability and interoperability of MAS components, Tran et al. [10] proposed a framework that supports the creation of methodologies supporting and making use of ontologies throughout much of the development lifecycle. To illustrate the use of the framework, an ontology based AOSE methodology, MOBMAS – a "*M*ethodology for *O*ntology-*B*ased *MAS*s" [11] was instantiated 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, organizational design and architecture design. MOBMAS takes advantage of existing AOSE methodologies, by reusing and enhancing their techniques and modelling definitions where appropriate. It endeavours to combine the strengths of the existing methodologies into one methodological framework [5, 11, 12].

In this paper, we overview MOBMAS [11, 13] and demonstrate its use for a community-based peer-to-peer (P2P) information sharing application. An experienced software developer who was not an author of the MOBMAS methodology was its user. The domain has been chosen for its current importance and it being a common application area of MAS. The case study shown in this paper is part of a wider comparative study to evaluate how various software developers respond to MOBMAS. After we overview MOBMAS and its use in sections 2 and 3, we conclude the paper with a discussion of limitations of the current verification of MOBMAS and future evaluation work to address some of the limitations.

2 MOBMAS Methodology

Using MOBMAS, 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. Each focuses on one of the following key area of MAS development: Analysis, Organization Design, Agent Internal Design, Agent Interaction Design and Architecture Design (Fig. 1). The development process of MOBMAS is highly iterative. MOBMAS activities are detailed in this section.

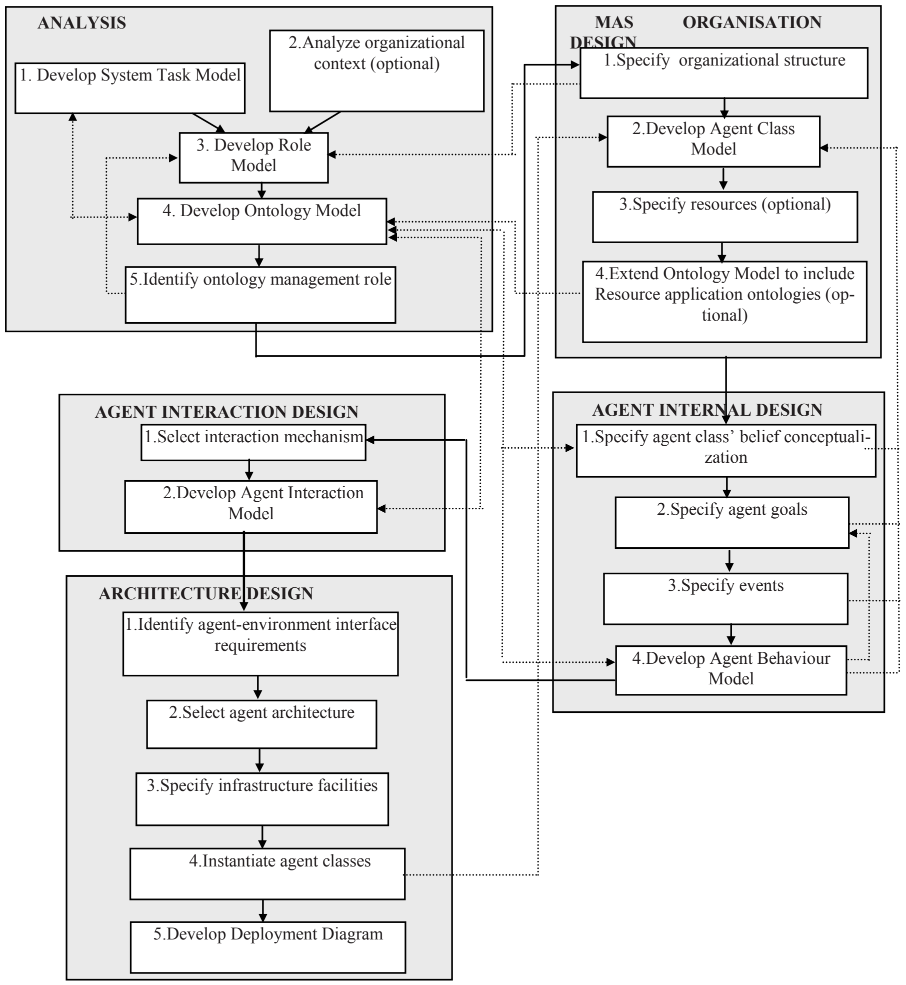
ANALYSIS
1. Develop System Task Model
2.Analyze organizational context (optional)
3. Develop Role Model
4. Develop Ontology Model
5.Identify ontology management role
MAS ORGANISATION DESIGN
1.Specify organizational structure
2.Develop Agent Class Model
3.Specify resources (optional)
4.Extend Ontology Model to include Resource application ontologies (optional)
AGENT INTERACTION DESIGN
1.Select interaction mechanism
2.Develop Agent Interaction Model
AGENT INTERNAL DESIGN
1.Specify agent class' belief conceptualization
2.Specify agent goals
3.Specify events
4.Develop Agent Behaviour Model
ARCHITECTURE DESIGN
1.Identify agent-environment interface requirements
2.Select agent architecture
3.Specify infrastructure facilities
4.Instantiate agent classes
5.Develop Deployment Diagram

Fig. 1. 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 square brackets.

2.1 Analysis Activity

The Analysis activity aims to form a conception for 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 the five following steps:

1 – Develop System Task Model: Takes as input specifications of the desirable functionality of the MAS and constructs a System Task Model showing system tasks, their functional decomposition, and any conflicts amongst them.

2 – Analyze Organizational Context (optional): Examines the human organization of the MAS and outputs an Organizational Context Model, which describes the existing structure of the contextual organization via Organization Context Charts.

3 – Develop Role Model: Identifies a set of roles that compose the MAS organization. MOBMAS offers various techniques for identifying roles from system tasks and from the existing organizational structure [11]. The Role Model captures all roles in the system, their associated tasks, acquaintances between roles and authority relationships governing inter-role acquaintances.

4 – Develop Ontology Model: Identifies and models the required ontologies to produce the Application Ontologies . These are basically a synthesis of Domain ontologies and Task ontologies that have been specialised to model the application's specific knowledge needs. MOBMAS offers some useful techniques for their development , although it refers the developers to the literature on ontology engineering for more support on this task.

5 – Identify Ontology-Management Role: Determines whether the ontology servers in the MAS will be freely accessed by agents, or controlled exclusively by a dedicated ontology manager.

The Role Model should be 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). It also specifies the ontological mappings between the MAS Application Ontologies.

2.2 MAS Organization Design

This activity refines the organizational structure of the target MAS and identifies a set of agent classes composing the MAS. If the MAS is a heterogeneous system that contains non-agent resources, these are also identified and their applications conceptualized. Four steps in this activity are as follows:

1 – Specify MAS Organizational Structure: Refines the preliminary organizational structure of MAS previously shaped by the Analysis Activity, by selecting an appropriate organizational style for the MAS (e.g. peer-to-peer or hierarchical), determining the authority relationships between roles. Two basic types of authority relationships are "peer" and "control". The Role Model should be updated to show these authority relationships between roles, and to include any new roles and/or inter-role acquaintances (e.g. Mediator or Broker role).

2 – Develop Agent Class Model: Identifies agent classes from roles. Generally, roles are assigned to agent classes via one-to-one mappings, but multiple roles can be mapped onto a single agent class for convenience, taking into account modularity and efficiency considerations. The developer should determine at design time whether the assignment of roles to agent classes is dynamic or static; that is, whether instances of an agent class may change their roles at run-time or not. In this sense, MOBMAS recognizes implicitly the notion of *'participation'* that ROADMAP makes explicit [14]. The Agent Class Model is depicted by Agent Class Diagrams and an Agent Relationship Diagram. The former defines each agent class in terms of its internal constructs (e.g. beliefs, goals, events), while the latter shows the acquaintances between agent classes, instantiation cardinality of each agent class, and the resources wrapped by agent classes (if any). Both Agent Class Diagrams and Agent Relationship Diagrams are developed in an incremental manner.

3 – Specify Resources (optional): Identifies and models non-agent resources that provide application-specific information and/or services to the agents, e.g. information sources or legacy systems. The output of this step is a Resource Model that describes each resource via a Resource Diagram. Two basic dimensions for describing a resource are its type and its Resource Application Ontology. The Resource Application Ontology is integrated in the Application Ontology developed during the Analysis Phase. The Agent Relationship Diagram should also be updated to show newly identified resources and their connections with wrapper agent classes.

4 – Revisit the Ontology Model: This updates, if required, the Ontology Model to add ontologies that conceptualize the resources' data, domains and/or services. Generally, each resource should be conceptualized by a separate Resource Application Ontology. The developer should also specify the mappings between Resource Application Ontologies and relevant MAS Application Ontologies, so as to enable the integration of these resources into the MAS application, as well as to support the interoperability between heterogeneous resources.

2.3 Agent Internal Design

For each agent class, this activity specifies belief conceptualization, agent goals, events, plan templates and reactive rules. It consists of four steps:

1 – Specify Agent Class' Belief Conceptualization: Identifies which (part of) application ontologies in the Ontology Model are needed by an agent class to conceptualize its run-time beliefs. These ontologies are those providing the necessary and relevant vocabulary for an agent instance to formulate and interpret factual knowledge about its world at run-time. The developer should update the Agent Class Diagram of each agent class to show the name of these application ontologies.

2 – Specify Agent Goals: Identifies the states of the world that an agent class aims to achieve or satisfy. Agent goals can be derived from role tasks as specified in the Role Model. Different agent classes may pursue an identical goal if they are mutually in charge of a joint role task (c-f. Section 2.2). The Agent Class Diagram of each agent class should be updated to list the identified goals.

3 – Specify Events: Identifies significant occurrences in the environment that agents need to respond to at run-time. These include events that activate agent goals and/or affect the agent's course of actions in fulfilling the agent goals. The Agent Class Diagram of each agent class should be updated to list the identified events.

4 – Develop Agent Behaviour Model: Specifies how each agent class behaves to achieve or satisfy each agent goal. We consider two behavioural style of agents: First is *planning behaviour*, where Agent Plan Templates are used to define the input information required by built-in planners to formulate actual plans for agents at run-time. These input information include: the target agent goal, potential sub-agent goals (if any), actions, events that may affect the agent's course of actions, commitment strategy and conflict resolution strategy. The second is *reactive behaviour* where Rule Specifications should be documented to specify the "if-then" rules linking the events and the required actions. Note that an agent class may adopt hybrid behaviour. The resulted Agent Behaviour Model should be validated for consistency against the Ontology Model and vice versa.

2.4 Agent Interaction Design

This activity models the interactions between agent instances, by selecting a suitable interaction mechanism for the target MAS and modelling the interactions. It has two steps: firstly, to decide upon which interaction mechanism is best suited to the target MAS. Two basic mechanisms considered by MOBMAS are "direct" (where agents directly exchange ACL messages following interaction protocols) and "indirect" (where agents indirectly exchange tuples). MOBMAS offers guidelines for selecting between these two mechanisms. Secondly, to define how agents interact depending on the selected interaction mechanism. In the direct interaction scheme, the developer should specify the interaction protocols governing the ACL message exchanges between agent instances. The Agent Interaction Model in this case is represented by a set of Interaction Protocol Diagrams. For the indirect interaction scheme, the developer should model the exchanges of tuples between agents and the shared tuplespace. In both cases, the developer should validate the Agent Interaction Model against the Ontology Model and vice versa, because the datatypes of all arguments in the ACL messages or tuples must be equivalent to the ontological concepts defined in the application ontologies. This requirement helps to ensure that the semantics of all information conveyed in ACL messages or tuples are consistently understood by the interacting agents. The Agent Class Model should also be checked to ensure that all communicating agent classes share the same application ontologies that govern their interactions. Lastly, the Agent Relationship Diagram should be updated to show descriptive information about each interaction pathway between agent classes.

2.5 Architecture Design

This activity deals with various design issues relating to agent architecture and MAS architecture. Its five steps produce different notational components that constitute the Architecture Model Kind. They are as follows:

1 – Identify Agent-Environment Interface Requirements: Investigates the requirements and characteristics of the agent perception, effect and communication mechanisms, so as to facilitate implementation of MAS. The requirements should be listed in an Agent-Environment Interface Requirement Specification.

2 – Select Agent Architecture: Decides upon the most appropriate architecture(s) for agents in the MAS. We suggest various factors to be considered when making the decision, e.g. the desirable style of agent behaviour, style of control, knowledge representation mechanism or scalability requirements. The selected agent architecture(s) should be graphically modelled in an Agent Architecture Diagram.

3 – Specify MAS Infrastructure Facilities: Identifies system components that are needed to provide system-specific services. All necessary infrastructure facilities should be listed in an Infrastructure Facility Specification.

4 – Instantiate Agent Classes: Determines the cardinality of instances in each agent class. This cardinality should be shown as an annotation next to the agent class name in the Agent Relationship Diagram.

5 – Develop MAS Deployment Diagram: Describes how the logical MAS architecture can be actuated in the operational environment. A MAS Deployment Diagram should be constructed to show the physical agent platforms, nodes, agent instances at each node and their acquaintances, and connections between nodes.

3 Community-Based P2P Information Sharing MAS

In this section, we illustrate the use of MOBMAS on a P2P information sharing application by an experienced system developer. The application and its specifications are based on Klampanos and Jose [15] and Mine et al.'s [16] conception of a P2P information sharing architecture.

3.1 Application Description

Each human user 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 (Fig. 2). 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).

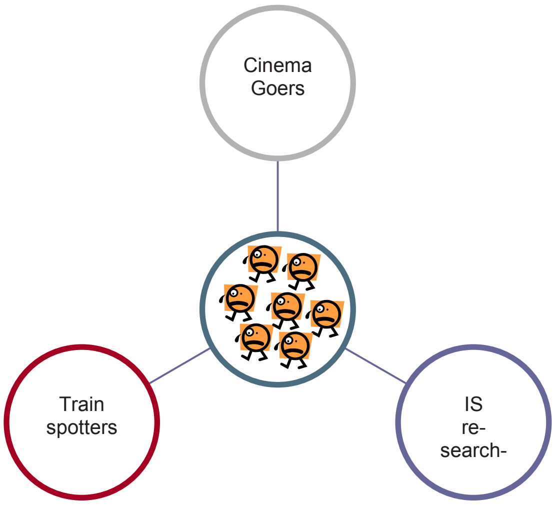

Fig. 2. The P2P Multi agent system is the collection of the agent assistants and any supporting specialized agents

As agents interact on behalf of their users, communities of interest begin to emerge. These communities may overlap (Fig. 3). 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.

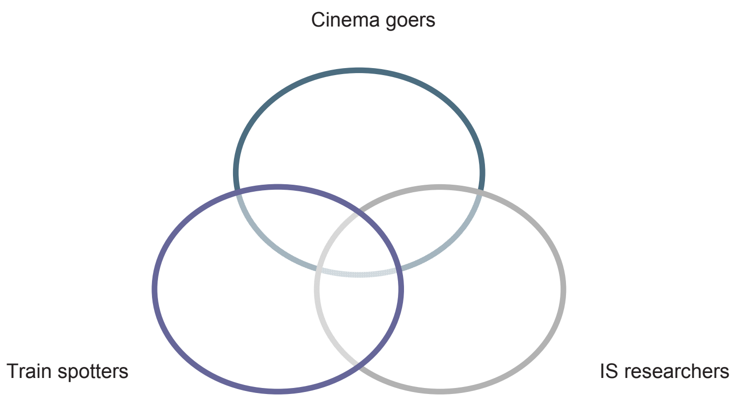

Fig. 3. A human user may belong to more than one community

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 behaviour 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 by 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 with1 additional information reflecting the interests of the agent that 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 updating 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 [16]. If no nodes can be short-listed, or if all candidate user-agents cannot provide the service required, the agent-user broadcasts the query to a wider circle of user-agents in the community, in order 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 first step of the Analysis activity was to identify the required system tasks and their functional decomposition (Figs. 4 and 5). The optional step in the Analysis activity to investigate the MAS organizational context is omitted in this application. A preliminary Role Model was developed to show tentative roles in the system and their tentative acquaintances. The developer identified roles by grouping closely-related system tasks in the System Task Model. For example, he assigned all the tasks dealing with interactions between the human user and his representing agents to a "User Interface" role (Fig. 8). The tasks relating to user query processing and file transferring (including file downloading and uploading) were 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".

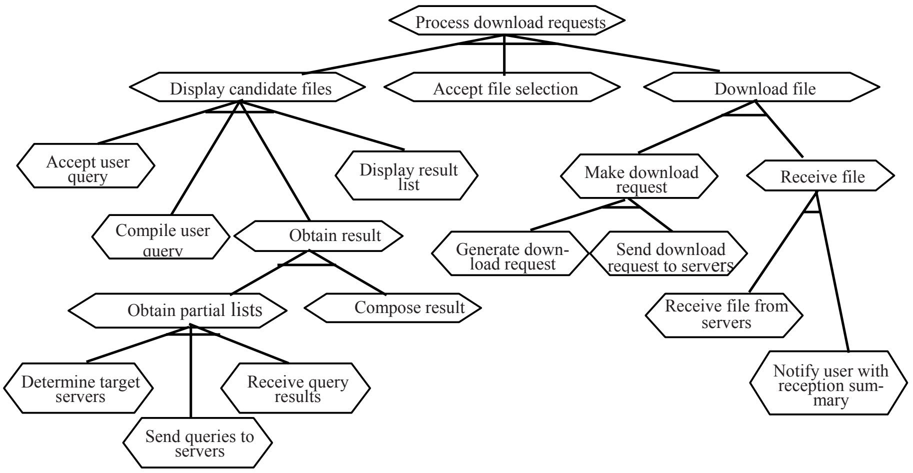

Fig. 4. System Task Diagram #1

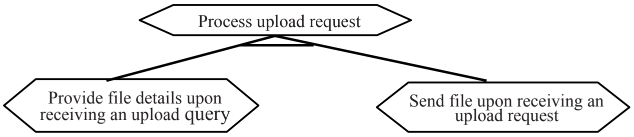
Process upload request
Provide file details upon receiving an upload query
Send file upon receiving an upload request

Fig. 5. System Task Diagram #2

An Ontology Model was then constructed to define the necessary application ontologies for the MAS. Only MAS Application Ontologies were initially examined, an ontology for conceptualising the information sharing domain is identified (Fig. 7). Resource Ontologies were identified later in the Organization Design activity (see Section 3.3) including an ontology for Movies information sharing (Fig. 6).

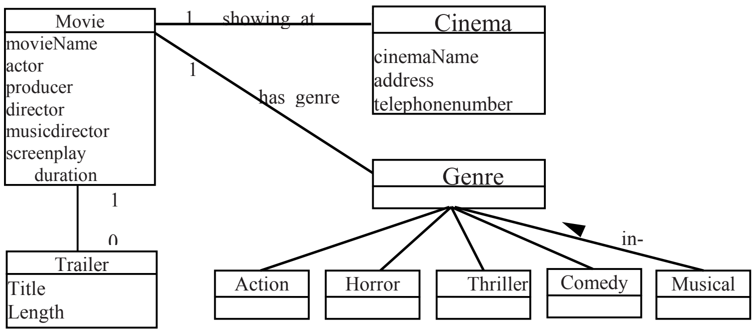

Fig. 6. Movie Ontology (from www.cse.dmu.ac.uk/~monika/Pages/ Ontolgies/)

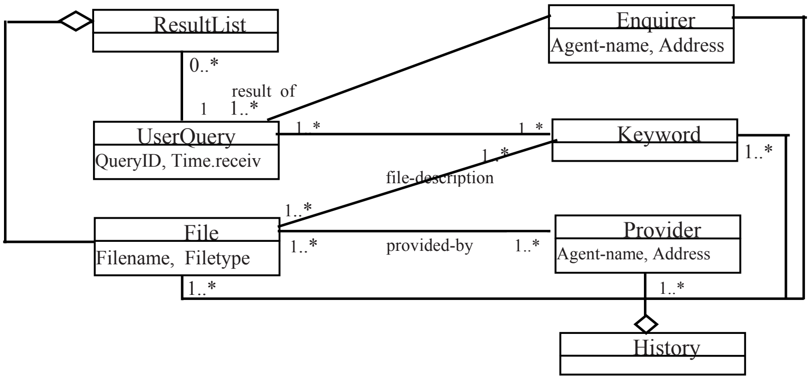

Fig. 7. Information Sharing Ontology

All agents in the system are expected to know about the two MAS Application Ontologies and they are not expected to change, so the developer decided that they should be stored at publicly-accessed ontology servers and can be accessed by all agents. No particular new role or agents was needed to manage and control these servers.

3.3 P2P MAS Organization Design

The first step in this activity was to refine the preliminary Role Model developed in Section 3.2 to specify authority relationships between roles (Fig. 8). Agent classes were then identified from roles. The Developer associated two roles "Searcher" and "Downloader" into one single agent class "Client".

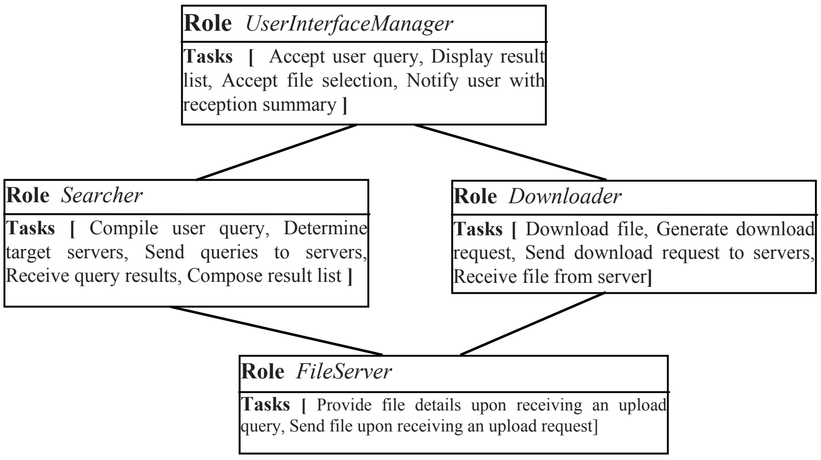
Role *UserInterfaceManager*
Tasks [Accept user query, Display result list, Accept file selection, Notify user with reception summary]
Role *Searcher*
Tasks [Compile user query, Determine target servers, Send queries to servers, Receive query results, Compose result list]
Role *Downloader*
Tasks [Download file, Generate download request, Send download request to servers, Receive file from server]
Role *FileServer*
Tasks [Provide file details upon receiving an upload query, Send file upon receiving an upload request]

Fig. 8. Role Diagram, all roles defined are in peer-to-peer relationships

Remaining roles were assigned to respective agent classes via one-to-one mappings. A preliminary Agent Relationship Diagram was constructed to show the tentative agent classes, their roles and their interaction pathways (Fig. 9). The Agent Class Diagram for each agent class was mostly empty at this stage, since no internal details were yet apparent. Note that the explicit separation of the "Client" and "Server" agent classes in the solution helped to clearly model that each user in the P2P network can be both a client and a server. Each user can be represented by a "Client" agent at one time, and by a "Server" agent some other times.

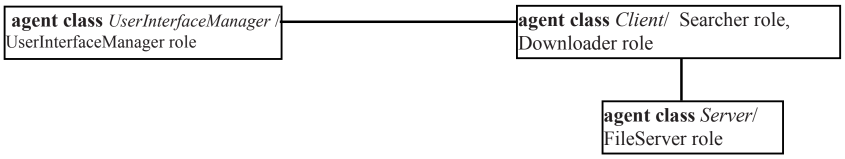
agent class *UserInterfaceManager* / UserInterfaceManager role
agent class *Client*/ Searcher role, Downloader role
agent class *Server*/ FileServer role

Fig. 9. Preliminary Agent Relationship

Non-agent software resources were identified: knowledge sources containing movie-related electronic files, e.g. web servers of HTTP files, directories of multimedia files. Each knowledge source needed 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 Role Model was extended to add a "Wrapper" role, and the Agent Class Model was updated to show the newly identified "Wrapper" agent class (Fig. 15).

The ontology conceptualizing each knowledge source was defined and thereafter added to the Ontology Model. Fig. 10 presents a Resource Application ontology for a knowledge source containing movie trailer files (which is one of the knowledge sources in the system) [and possibly many other domain ontologies for information sharing in a fully fledged deployed P2P system]. Mappings between concepts of this ontology (named "MovieTrailer Resource Ontology") and the "Movie Ontology" were also specified. In Fig. 10, the "Movie Ontology" is re-shown in grey (c.f. Fig. 6). Relationships annotated with predicate "equivalent" represent the semantic mappings between the concepts of "MovieTrailer Resource Ontology" and those of "Movie Ontology". Note that other knowledge sources in the system would have their own resource ontologies and would be mapped differently to the "Movie Ontology".

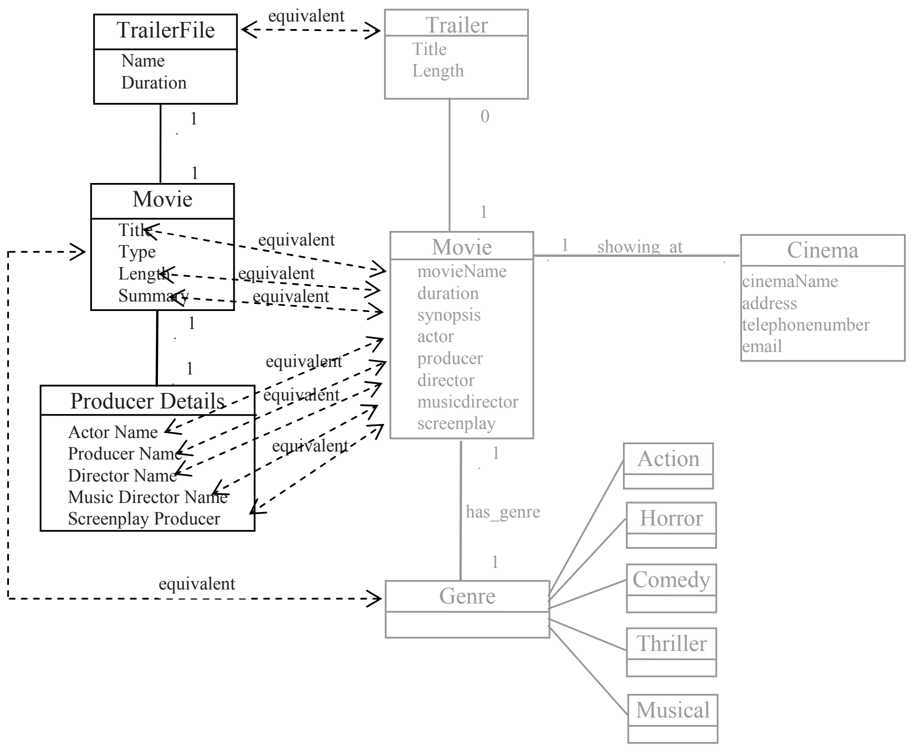

Fig. 10. MovieTrailer Resource Ontology

3.4 P2P Agent Internal Design

The internal design of each agent class begins with the identification of ontologies conceptualizing the agent's run-time beliefs. Agent goals are identified directly from role tasks. However, while role tasks were specified using imperatives, agent goals were specified as "something is achieved". For instance, the task "Provide file details upon receiving an upload query" of the role "FileServer" (Fig. 8) revealed an agent goal "File details are provided upon receiving an upload query".

Events affecting agents' courses of actions were also identified. The Agent Class Model, particularly the Agent Class Diagrams, was then updated to show the listing of belief conceptualization, agent goals and events for each agent class. Fig. 11 illustrates the updated Agent Class Diagram for the "Server" agent class.

agent class *Server* / FileServer role
belief conceptualization Movie Ontology Information Sharing Ontology MovieTrailer Resource Ontology
agent-goals File details are provided upon receiving an upload query File is sent upon receiving an upload request
Events Reception of upload query Reception of upload request

Fig. 11. Agent Class Diagram for the "Server" agent class

An Agent Behaviour Model was developed to define plan templates and reactive rules for each agent class. The developer considered both planning and reactive behaviour for each agent class, depending on the target agent goal. For example, the "Server" agent class required planning behaviour to fulfil the agent goal "File details are provided upon receiving an upload query". The Agent Plan Template and Agent Plan Diagram for these behaviours is illustrated (Figs. 12 and 13 respectively). The Agent Behaviour Model was also validated against the Ontology Model and Agent Class Diagram.

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

Fig. 12. Agent Plan Template for the "Server" agent class

3.5 P2P MAS Agent Interaction Design

The first step here was to select a suitable interaction mechanism for the MAS. The developer chose the direct interaction mechanism using ACL, rather than "indirect" mechanism because he believed that the speech-act performatives provided by ACL can better support the high level of communication semantics required by this application than primitives provided by an indirect mechanism. The target application can also reuse many interaction protocols provided by existing catalogues, such as FIPA's Protocol Library [17]. An Agent Interaction Model was then developed to define interaction protocols between agents. Each protocol was represented by an AUML Interaction Protocol Diagram (Fig. 14). The developer also checked the Agent Interaction Model against the Ontology Model for consistent use of constructs.

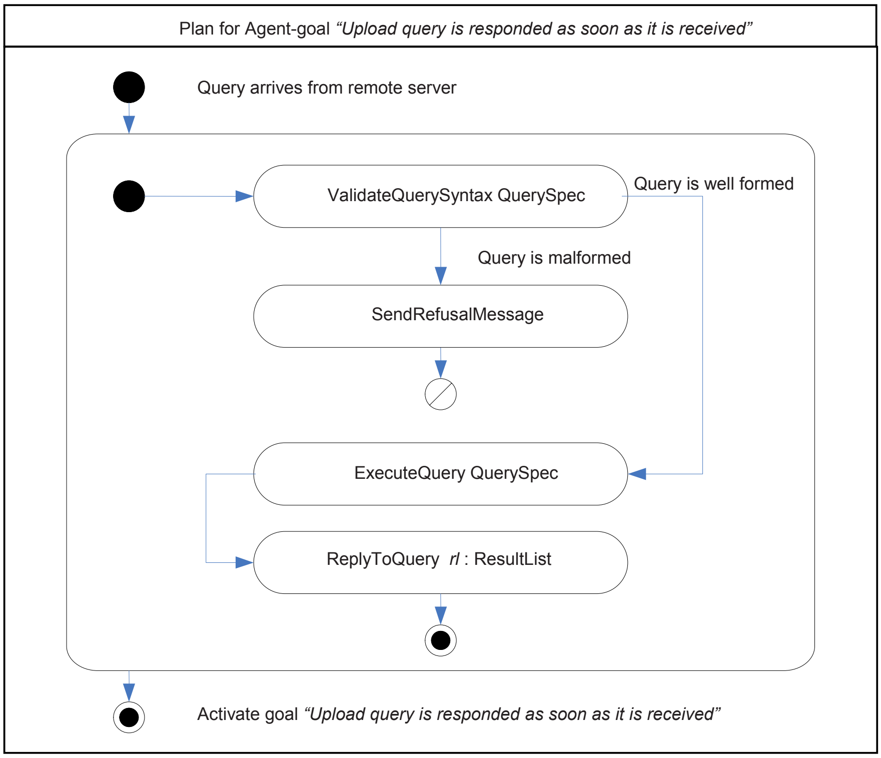
Plan for Agent-goal *"Upload query is responded as soon as it is received"*
Query arrives from remote server
ValidateQuerySyntax QuerySpec
Query is well formed
Query is malformed
SendRefusalMessage
ExecuteQuery QuerySpec
ReplyToQuery *rl* : ResultList
Activate goal *"Upload query is responded as soon as it is received"*

Fig. 13. Agent Plan Diagram (for the *Server* agent class)

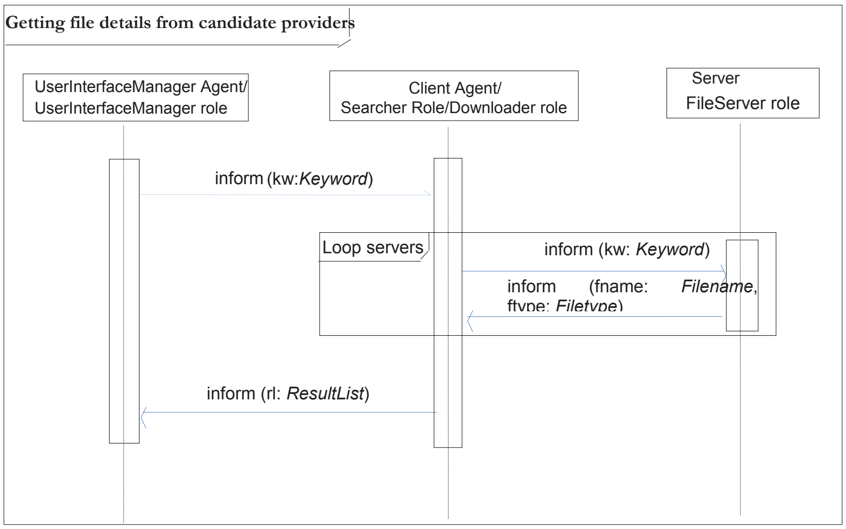
Getting file details from candidate providers
UserInterfaceManager Agent/
UserInterfaceManager role
Client Agent/
Searcher Role/Downloader role
Server
FileServer role
inform (kw:*Keyword*)
Loop servers
inform (kw: *Keyword*)
inform (fname: *Filename*, ftype: *Filetype*)
inform (rl: *ResultList*)

Fig. 14. Interaction Protocol Diagram

The Agent Relationship Diagram gets finally updated to show descriptive information for each interaction pathway between agent classes specifying the Interaction Protocol and the used ontology.

4 Discussion, Limitations and Future Work

We have demonstrated the first four phases of MOBMAS – a methodology for ontology-based MAS development – on a P2P community-based information sharing application. (Due to space restrictions the results of the last phase, MAS Architecture Design are not reported here). This work is part of an on-going evaluation of MOBMAS as reported in [13]. The MOBMAS methodology was initially reviewed and refined based on the feedback of two experts in AOSE. The refined methodology was then used by two different developers to design a peer to peer community-based information sharing application (this paper reports the results from one developer; the results from the other developer are reported in [18]). 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 the conducted on the final version of MOBMAS to verify MOBMAS' ability to support important AOSE methodological features, steps and modelling concepts.

In both case studies the use of MOBMAS was conducted by experienced software developers, who were not authors of the MOBMAS methodology but was given detailed documentation of the methodology. Both case studies included developers responses to a detailed questionnaire regarding the usage of MOBMAS. Existing surveys indicate that they valued the step-by-step development process of MOBMAS and the provision of many heuristics and techniques to support each step.

It must noted that even with the set of case studies and the questionnaires, without comparing MOBMAS against other methodologies 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. 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 Behaviour 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 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. Beydoun et al. recently in [19] completed a preliminary framework which checks the early requirements against role models. This will be usable as a stepping stone to develop a supporting tool (as suggested by one of the reviewers).

References

1. Gruber, T.R.: Automated Knowledge Acquisition for Strategic Knowledge. Machine Learning 4, 293–336 (1989)
2. Shave, M.J.R.: Ontological Structures for Knowledge Sharing. New Review of Information Networking 3, 125–133 (1997)
3. Uschold, M., Grueninger, M.: Ontologies: Principles, Methods and Application. Knowledge Engineering Review 11, 93–195 (1996)
4. Fensel, D.: Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. Springer, Heidelberg (2001)
5. Tran, Q.N., Low, G.: Comparison of Methodologies. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, pp. 341–367. Idea Group Publishing, USA (2005)
6. Girardi, R., Faria, C.G.: Ontology-based Domain Modelling of Multi-Agent Systems. In: OOPLSA Workshop, pp. 295–308 (2004)
7. Dileo, J., Jacobs, T., Deloach, S.: Integrating Ontologies into Multi-Agent Systems Engineering. In: AOIS2002. 4th International Bi-Conference Workshop on Agent Oriented Information Systems, Italy (2002)
8. Brandao, A.A.F., Silva, V.T.d., Lucena, C.J.P.d.: Ontologies as Specification for the Verification of Multi-Agent Systems Design. In: Object Oriented Programmings, Systems, Languages and Applications Workshop (2004), California (2004)
9. Esteva, M.: Electronic Institutions: From Specification To Development. In: UAB - Universitat Autonòma de Barcelona, Barcelona. Artificial Intelligence Research Insitute (2003)
10. Tran, Q.N., Low, G., Beydoun, G.: A Methodological Framework for Ontology Centric Agen oriented Software Engineering. Computer Science Systems and Engineering 21, 117–132 (2006)
11. Tran, N.: MOBMAS: A Methodology for Ontology-Based Multi-Agent Systems Development. In: School of Information Systems, Technology and Management, Vol. PhD. UNSW, Sydney (2005)
12. Tran, Q.N.N., Low, G.C., Williams, M.A.: A preliminary comparative feature analysis of multi-agent systems development methodologies. In: Bresciani, P., Giorgini, P., Henderson-Sellers, B., Low, G., Winikoff, M. (eds.) AOIS 2004. LNCS (LNAI), vol. 3508, pp. 157–168. Springer, Heidelberg (2005)
13. Tran, Q.N.N., Low, G.C.: MOBMAS: A Methodology for Ontology-Based Multi-Agent Systems Development. Information and Software Technology (2007)

14. Juan, T., Pearce, A.R., Sterling, L.: ROADMAP: extending the gaia methodology for complex open systems. In: First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2002), Italy (2002)
15. Klampanos, I.A., Jose, J.M.: An Architecture for Peer-to-Peer Information Retrieval. In: SIGIR 2003, pp. 401–402 (2003)
16. Mine, T., Matsuno, D., Kogo, A., Amamiya, M.: Design and Implementation of Agent Community Based Peer-to-Peer Information Retrieval Method. In: Klusch, M., Ossowski, S., Kashyap, V., Unland, R. (eds.) CIA 2004. LNCS (LNAI), vol. 3191, pp. 31–46. Springer, Heidelberg (2004)
17. FIPA: FIPA Query Interaction Protocol Specification (2003)
18. Tran, N., Beydoun, G., Low, G.C.: Design of a Peer-to-Peer Information Sharing MAS Using MOBMAS (Ontology-Centric Agent Oriented Methodology). In: Information Systems Development (ISD 2006), Budapest (2006)
19. Beydoun, G., Krishna, A.K., Ghose, A., Low, G.C.: Towards Ontology-Based MAS Methodologies: Ontology Based Early Requirements. In: Information Systems Development Conference (ISD 2007), Galway (2007)