Comparing Agent-Oriented Methodologies

Khanh Hoa Dam
kdam@cs.rmit.edu.au

Michael Winikoff
winikoff@cs.rmit.edu.au

School of Computer Science and Information Technology
RMIT University, Melbourne, Australia

ABSTRACT

Numerous methodologies for developing agent-based systems have been proposed in the literature. However, their application is still limited due to their lack of maturity. Evaluating methodologies' strengths and weaknesses plays an important role in improving them and in developing the "next-generation" of methodologies. This paper presents a comparison of three prominent agent-oriented methodologies: **MaSE**, **Prometheus** and **Tropos**. It is performed based upon an attribute-based framework which addresses four major areas: *concepts*, *modelling language*, *process* and *pragmatics*. The objectivity of the comparison is increased by including inputs from the authors of the methodologies using a questionnaire and by conducting an experimental evaluation of the methodologies.

1. INTRODUCTION

"*One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems.*" [14, page 11].

Even though many Agent Oriented Software Engineering (AOSE) methodologies have been proposed, few are mature or described in sufficient detail to be of real use. We believe that the area of agent-oriented methodologies is maturing rapidly and that the time has come to begin drawing together the work of various research groups with the aim of developing the "next generation" of agent-oriented software engineering methodologies.

A crucial step is to understand the relationship between the various key methodologies, and particularly to understand each methodology's strengths, weaknesses, and domains of applicability. In this paper we perform the comparison on several well-known methodologies. These were selected since they (a) were described in more detail (e.g. journal paper rather than a conference paper); and (b) were perceived as being significant by the agent community. Another important factor was whether the methodology had been developed over an extended time period based on feedback from users other than the developers of the methodology. Based on these criteria we chose the five[1] methodologies Gaia [26, 27], MESSAGE [4, 3], MaSE [8, 7], Prometheus [16, 17, 18, 19] and Tropos [1, 11]. However due to space limitations only **MaSE**, **Prometheus** and **Tropos** are presented in this paper[2].

In section 2, we briefly introduce these methodologies. Since it is impossible to accurately summarise a detailed methodology in a single page we attempt to give a flavour of each methodology, and to outline the process and notations used. We refer the reader to original sources for further details on each methodology.

We then (section 3) describe a framework for comparing AOSE methodologies and, in section 4, apply the framework to compare the methodologies.

In doing this there are two key issues. Firstly, how can we ensure that the framework is unbiased and complete (covers all significant criteria); and secondly, how can we avoid the comparison being affected by our biases? In particular, one of the authors of this paper is also one of the developers of the Prometheus methodology.

The first issue is addressed by basing the framework on existing work in the comparison of Object-Oriented (OO) methodologies. By including a range of issues that have been identified as important by a range of authors we avoid biasing the comparison by including only issues that we consider important. The second issue we address by having others do the assessment of each methodology. Specifically, for each methodology we asked the authors of the methodology to fill in a questionnaire assessing the methodology. We also had students each develop designs for the same application using different methodologies. We collected comments from the students as they developed their application designs over summer (Dec. 2002 – Feb. 2003) as well as asking them to fill in the questionnaire. The aim of the questionnaire was *not* to provide a statistically significant sample in order to carry out a scientific experiment. This was not practical. Rather, the aim was to avoid any particular bias by having a range of viewpoints.

The experimental application was a Personal Itinerary Planner System (PIPS). Its main goal is to assist a traveller in finding activities. Suppose you are visiting an unfamiliar city and have nothing planned on a weekend or evening. Rather than spend a day wondering aimlessly (or worse, working in a hotel room), you access PIPS. After telling

[1]We were limited to choosing five methodologies since we had that many summer studentships. This prevented us from being able to compare all of the methodologies that met the selection criteria.

[2]We did not receive questionnaire responses from the creators of Gaia which prevented us from selecting it as one of the three methodologies covered in this paper.

PIPS where you are, when you are looking for things to do, and what are your interests it responds with a number of itineraries. An itinerary consists of one or more activities such as eating at a restaurant, going to a show, visiting a local attraction (e.g. zoo, aquarium, historic site, etc.). However it is not just a collection of activities. The itinerary should meet constraints on time and space. Where an activity is followed by another activity at a different location they should be separated by a transport activity that takes the user from the location of the first activity to the location of the second.

2. THE METHODOLOGIES

2.1 MaSE

The goal of *Multiagent Systems Engineering* (MaSE) [8] is to provide a complete-lifecycle methodology to assist system developers to design and develop a multi-agent system. It fully describes the process which guides a system developer from an initial system specification to system implementation. This process consists of seven steps, divided into two phases.

The MaSE **Analysis** stage includes three smaller process steps. First, the *Capturing Goals* step guides the analysts to identify goals and structure and represent them as a *goal hierarchy*. The second step, *Applying Use Cases*, involves extracting main scenarios from the initial system context or copying them from it if they exist. These use cases are also used to build a set of *sequence diagrams* (similar to UML sequence diagrams). *Refining Roles* is the final step of the Analysis phase where a *Role Model* and a *Concurrent Task Model* are constructed. The Role Model describes the roles in the system. It also depicts the goals which those roles are responsible for, the tasks that each role performs to achieve its goals and the communication path between the roles. Tasks are then graphically represented in fine-grained detail as a series of finite machine automata in the Concurrent Task Model.

The first step of the **Design Phase** is called "*Creating Agent Classes*". The output of this step is an *Agent Class Diagram* which describes the entire multi-agent system. The agent class diagram shows agents and the roles they play. Links between agents show conversations and are labelled with the conversation name. For example, in figure 1 the agent AccountWatcher comprises two roles (AccountManager and AccountVerifier) and it converses with the UserInterface agent type. The details of the conversations are described in the second step of the design phase ("*Constructing Conversations*") using *communication class diagrams*. These are a form of finite state machine. The third step of the Design stage is *Assembling Agent Classes*. During this step, we need to define the agent architecture and the components that build up the architecture. In terms of agent architecture, MaSE does not dictate any particular implementation platform. The fourth and final step of the design phase is *System Design*. It involves building a *Deployment Diagram* which specifies the locations of agents within a system.

MaSE has extensive tool support in the form of agentTool [7]. Its latest version 2.0[3] implements all seven steps of MaSE. It also provides automated support for transforming

[3]http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm

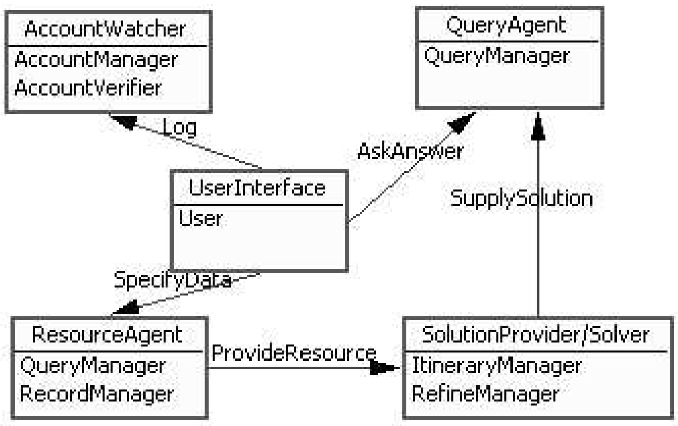

Figure 1: PIPS agent class diagram (extracted from the PIPS design documentation produced by Yenty Frily using agentTool)

analysis models into design constructs.

2.2 Prometheus

The *Prometheus* [16, 17, 18, 19] methodology is a detailed AOSE methodology that is aimed at non-experts. It has been successfully taught to and used by undergraduate students. Prometheus consists of three phases: *system specification*, *architectural design* and *detailed design*.

The first phase of Prometheus, the **system specification** phase, involves two activities: *determining the system's environment*, and *determining the goals and functionality of the system*. The system's environment is defined in terms of *percepts* (incoming information from the environment) and *actions* (the means by which an agent affects its environment). In addition external data is defined. Defining the system's functionality is done by identifying goals, identifying functionalities that achieve these goals, and by defining use case scenarios. Use case scenarios describe examples of the system in operation. A typical scenario includes a sequence of steps depicting incoming percepts, messages sent and the activities and actions. The development of goals, functionalities and use case scenarios is usually iterative. Each of the concepts is captured using a descriptor form.

The second stage, **architectural design**, involves three activities: *defining agent types*, *designing the overall system structure*, and *defining the interactions between agents*. Agent types are derived by grouping functionalities. The choice of grouping is guided by consideration of coupling and cohesion which are identified with the aid of the data coupling diagram and agent acquaintance diagram. Each identified agent type is described using an agent descriptor which describes the lifecycle of this agent type (how and when it is initialized and destroyed), its functionalities, the data it uses and produces, its goals, the events it should respond to, its actions and the other agent types that it interacts with. The system's structure is captured in a *system overview diagram*, arguably the single most important design artifact in Prometheus. The system overview diagram provides the designers and implementers a general picture of how the system as a whole will function. It shows the agent types, the communication links between them, and

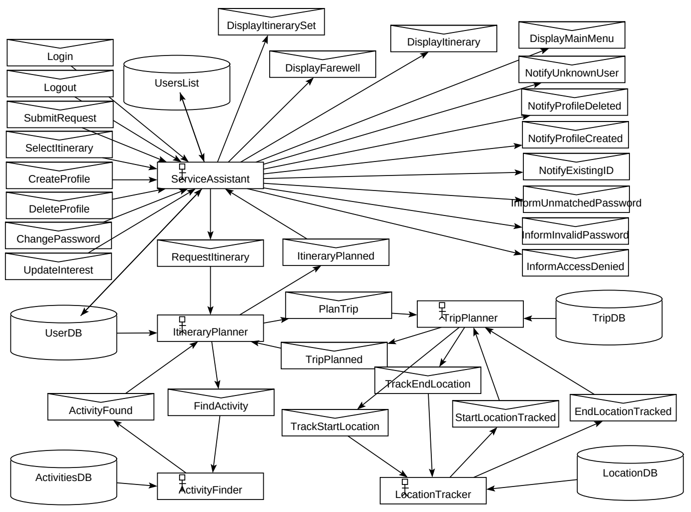

Figure 2: PIPS System Overview (extracted from the PIPS design documentation produced by Robert Tanaman using the Prometheus Design Tool (PDT))

data. It also shows the system's boundary and its environment (actions, percepts, and external data). An example of PIPS's system overview diagram is shown in Figure 2. Whereas the system overview diagram captures the static structure of the system, interaction protocols capture the dynamic behaviour of the system by defining the intended valid sequences of messages between agents. The interaction protocols are developed from interaction diagrams which in turn are based on the scenarios.

The internals of each agent and how it will accomplish its tasks within the overall system are addressed in the **detailed design** phase. It focuses on *defining capabilities*, *internal events*, *plans* and *detailed data structure* for each agent type identified in the previous step. Firstly, an agent's capabilities are depicted via a capability descriptor which contains information such as which events are generated and which events are received. The capability descriptor also includes a description of the capability, details involving interactions with other capabilities and references to data read and written by the capability. Secondly, at a lower level of detail, there are other types of descriptors: individual plan descriptors, event descriptors, and data descriptors. These descriptors provide the details so that they can be used in the implementation phase. The detailed design phase also involves constructing agent overview diagrams. These are very similar to the system overview diagram in terms of style but give the top level view of each agent's internals rather than the system as a whole. Agent overview diagrams, together with the capability descriptors, provides a high level view of the components within the agent internal architecture as well as the their connectors (interactions). They show the top level capabilities of the agent, the flow of tasks between these capabilities and data internal to the agent.

Prometheus is supported by two tools. The JACK Development Environment (JDE), developed by Agent Oriented Software (*www.agent-software.com*) includes a design tool that allows overview diagrams to be drawn. These are linked with the underlying model so that changes made to diagrams, for example adding a link from a plan to an event, are reflected in the model and in the corresponding JACK code. The Prometheus Design Tool (PDT) provides forms to enter design entities. It performs cross checking to help ensure consistency and generates a design document along with overview diagrams. Neither PDT nor the JDE currently support the system specification phase.

2.3 Tropos

Tropos [1, 11] is an agent-oriented software development methodology created by a group of authors from various universities in Canada and Italy. One of the significant differences between Tropos and the other methodologies is its strong focus on early requirements analysis where the domain stake-holders and their intentions are identified and analysed. This analysis process allows the reason for developing the software to be captured. The software development process of Tropos consists of five phases: *Early Requirements*, *Late Requirements*, *Architectural Design*, *Detailed Design* and *Implementation*.

Early Requirements: The requirements phase of Tropos is influenced by Eric Yu's i* modelling framework [28]. Tropos uses the concept of actor and goals to model the stake-holders in the target domain and their intentions respectively. Tropos divides goals into two different types. Hardgoals eventually lead to functional requirements whilst softgoals[4] relate to non-functional requirements. There are two models that represent goals and actors at this point in the methodology. First, the actor diagram depicts the stake-holders and their relationship in the domain. The latter are called "social dependencies" that reflect how actors depend on one another for goals to be accomplished, plans to be executed, and resources to be supplied. Second, the goal diagram shows the analysis of goals and plans with regard to a specific actor who has the responsibility of achieving them. Goals and plans are analysed based upon several reasoning techniques as proposed by the methodology such as: means-end analysis, AND/OR decomposition, and contribution analysis. These techniques help the analysts structure the system's goals, identify softgoals, plans and resources providing the means for accomplishing a goal, and capture goals that promote or interfere with the fulfilment of other goals.

Late Requirements: This phase involves extending the models which were created in the previous step. The importance of this stage is the modelling of the target system within its environment. The system-to-be is modelled as one or more actors. Its interdependencies with other actors in the models contribute to the accomplishment of stake-holder goals. Therefore, these dependencies define the target system's functional and non-functional requirements. Figure 3 show the dependency of Tourism Commission on PIPS to provide information (hard goal). It also requires a usable PIPS (softgoal). These goals are then decomposed into sub-goals. For instance, the goal "provide information" is fulfilled by the composite achievement of two sub-goals "search information" and "generate output". The subgoal "search information" in turn has several sub-goals such as "access to tourism database", and "access to user's records". Additionally, the positive contribution of other goals to the softgoal

[4]Softgoals are goals whose satisfaction conditions cannot be precisely defined.

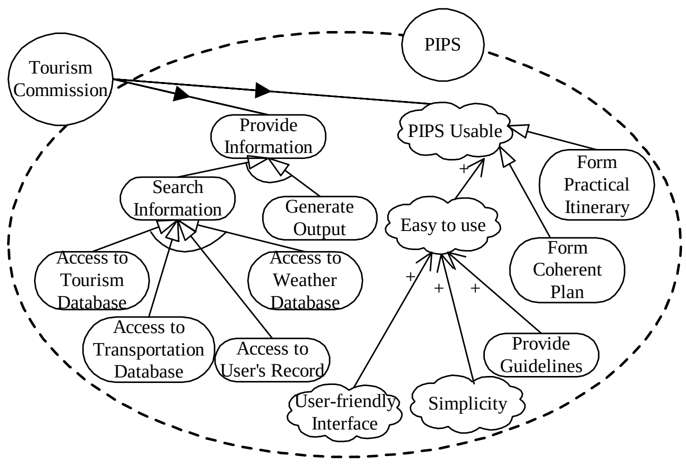

Figure 3: Goal Diagram – PIPS in connection with Tourism Commission (re-drawn based on the PIPS design documentation produced by Sindawati Hoetomo)

"easy to use" is also shown. A "user-friendly interface" that offers "simplicity" and provides guidelines promotes the fulfilment of the goal "easy to use".

Architectural Design: Tropos defines three steps which system designers can apply to proceed through this phase. At the first step, new actors are included and described by an extended actor diagram. These new actors are derived based on the choice of architectural style. They may exist to fulfil non-functional requirements or to support sub-goals decomposed in the previous steps. The second and third steps respectively identify the capabilities and group them to form agent types, where each agent types is formed by joining some number of capabilities.

Detailed Design: The Tropos detailed design phase involves defining the specification of agents at the micro level. There are three different types of diagrams which the designers need to produce to depict the capabilities, the plans of agents and the interactions between them. Tropos uses UML activity diagrams to represent capabilities and plans at the detailed level. Plan diagrams are fine-grained representations of each plan node in the capability diagrams. The interaction between agents in the system is represented by agent interaction diagrams. They are in fact AUML interaction diagrams.

Implementation: Having finished the detailed design stage, we can now move to the final step of Tropos, the Implementation phase. Tropos chooses a BDI platform, specifically JACK Intelligent Agents$^{(\mathrm{TM})}$ [2], for the implementation of agents. JACK provides five main language constructs: agents, capabilities, database relations, events and plans. At this stage, developers need to map each concepts in the design phase to the five constructs in JACK. Tropos provides several guidelines and heuristics for mapping Tropos concepts to BDI concepts and BDI concepts to JACK constructs.

3. A COMPARISON FRAMEWORK

In this section, we briefly describe a methodology evaluation framework within which the methodology comparison is conducted. The framework consists of a set of criteria which addresses not only classical software engineering attributes but also properties which are uniquely found in AOSE. In order to avoid using an inappropriate comparison framework the properties in our framework are derived from a survey of work on comparing AOSE methodologies [5, 15, 23, 24], and more importantly on comparing OO methodologies [9, 10, 12, 21, 22, 25]. The comparison framework covers four major aspects of each AOSE methodology: **Concepts**, **Modelling language**, **Process** and **Pragmatics**. This framework is adapted from a framework proposed in [9] for comparing Object-Oriented Methodologies. In addition to the four above components, that framework considers two more areas: Support for Software Engineering and Marketability. For our framework, we have decided to address "Support for Software Engineering" criteria in various places in the above four major aspects. With regard to "Marketability" issues, since all of our compared AOSE methodologies are still being researched and developed we do not believe that marketability criteria are applicable.

Concepts: Agent-oriented concepts are of great importance for agent-oriented methodologies in general and for agent-oriented modelling languages in particular. Based on the literature research, we have found a set of significant agent-oriented concepts that are commonly addressed. These include the definition of agents, their characteristics such as autonomy, adaptability, mental notions (such as beliefs, desires and intention), the relationship and communication between agents, and other concepts such as goals, agent roles and capabilities, as well as percepts, actions and events. When reviewing each AOSE methodology's definition of those agent-oriented concepts, we essentially look at the extent to which the methodology supports the concept or to which it supports the construction of agents that posses the attribute.

Modelling language: If agent-oriented concepts are the basis for any AOSE methodology, then the modelling language for representing designs in terms of those concepts is generally the core component of any software engineering methodology. A typical modelling language consists of three main components [10]: symbols (either graphical or textual representation of the concepts), syntax and semantics. It is important that the modelling language allows the system under development to be modelled from different views such as behavioural, functional and structural views [25]. As a result, by having a good modelling notation the methodology effectively eases the complex tasks of requirement analysis, specification analysis and design. Therefore, measuring the quality of the modelling language of an AOSE methodology plays an important part in our evaluation.

The criteria which assess the modelling language of each methodology are categorised into two groups. *Usability* criteria reflect usage requirements of a modelling language in terms of providing a means for software developers to exchange their thoughts and ideas. These criteria basically address the question of how easy the notation and the models are to understand and to use [21, 22]. They address the complexity, clarity and understandability of a modelling language. In addition we also assess whether the modelling notation is adequate for expressing all the necessary concepts and whether it is expressive, that is whether these concepts are expressed in a natural and direct way.

The second group of criteria to assess a modelling language is *technical criteria*. They involve the **unambiguity** and **consistency** of a modelling language. **Unambiguity** means that a constructed model can be interpreted unambiguously whereas **consistency** is a technical quality relating to the assistance of a modelling technique to the software designer in guaranteeing that between representations, no set of individual requirements is in conflict [25]. The technical qualities of a modelling language also concern its ability to support **traceability**. This is the ability to track dependencies between different models and between models and code.

Process: As discussed above, the modelling language is considered as a mandatory part of any software engineering methodology. However, in constructing a software system, software engineering also emphasizes the series of activities and steps performed as part of the software life cycle [9, 12, 25]. These activities and steps form the process which assists system analysts, developers and managers in developing software. According to [9], an ideal methodology should cover six stages, which are enterprise modelling, domain analysis, requirements analysis, design, implementation and testing. Methodologies which cover all aspects of the system development are more likely to be chosen because of the consistency and completeness they provide.

An important aspect in evaluating whether a methodology covers a particular process step is the degree of detail provided. It is one thing to mention a step ("at this point the designer should do X") and another thing to provide a detailed description of *how* to perform X. Since design is inherently about tradeoffs, detailed descriptions are usually expressed using heuristics rather than algorithms, as well as examples. Thus, in assessing support for process steps we identify whether the step is mentioned, whether a process for performing the step is given, whether examples are provided, and whether heuristics are given. In addition, it is necessary to consider which development contexts are supported by the methodology. In particular, we need to examine whether a particular AOSE methodology supports legacy system integration. This criterion is important, especially for AOSE because, as emphasized in [13], one of the key pragmatic issues which determine whether the agent-oriented paradigm can be popular as a software engineering paradigm is the degree to which existing software can be integrated with agents. In addition to heuristics, the availability of estimating guidelines is essential in aiding the project planning tasks. Hence, we also investigate whether the assessed methodology provides estimating guidelines such as: the estimating the costs, schedule, etc. of the developed system.

Pragmatics: In addition to issues relating to notation and process, the choice of an AOSE methodology depends on the *pragmatics* of the methodology. This can be assessed based on two aspects [15]: management and technical issues. Management criteria should consider the support that a methodology provides to management when adopting it. They include the cost involved in selecting the new methodology, its maturity and its effects on the current organization business practices [9, 15, 25]. There are different types of cost associated with adopting the methodology such as the cost of acquiring methodology and tool support, and the required training to fully exploit the methodology. The methodology maturity, on the other hand, concerns the resources available to support the methodology (e.g. documentation, training, consulting services, etc.), and the availability of automated tools. In addition, the history of the methodology's use is considered so that a methodology that has been used to create industrial strength applications should be preferred over ones that have only been used to develop small demonstration projects.

Differing from management issues, technical criteria look at a methodology from another angle. They consider whether the methodology is targeted at a specific type of software domain such as information systems, real time systems or component-based systems [15]. With regard to this issue, the methodology that is applicable to a wide range of software domains tends to be more preferred. Additionally, technical evaluation considerations measure the methodology's support for designing systems that are scalable. It means that the system should allow the incorporation of additional resources and software components with minimal user disruption.

4. COMPARING METHODOLOGIES

Based on the above comparison framework, we have developed a questionnaire[5] which consists of around 60 questions addressing the concepts, modelling language, process and pragmatics of a methodology. The questionnaire was distributed to the authors of each AOSE methodology which we have selected to compare (MESSAGE, Gaia, MaSE, Tropos, and Prometheus). 12 responses (out of 16) have been received.

In addition to obtaining evaluations from the authors of the methodologies, we also had a number of students who, over the summer, designed an agent application, each using a different methodology. Each student gave us feedback on their experience in understanding and using the methodology, and also completed the questionnaire at the end of their work. For each of the three methodologies both the creators of the methodology responded to the questionnaire as did the users (summer students). The results are summarised in figure 4 and are discussed below.

Concepts: With regard to agent-oriented concepts, the level of support for autonomy of all of the methodologies is overall good (ranging from medium to high). Prometheus & Tropos support very well the use of mental attitudes (such as beliefs, desires, intentions) in modelling agents' internals (medium to high) whereas MaSE provides weaker support. The questionnaire also addresses the support for pro-activeness and reactiveness, however it seems that these two attributes are difficult to measure even though they seem to be fairly well supported by all three methodologies (medium-high for MaSE and Prometheus, mostly high for Tropos). In terms of support for concurrency, although the ratings are mostly medium-high and varied considerably, MaSE is probably best with its protocol analyser, and Prometheus was rated as being one of the weakest[6]. Although the methodologies all support cooperating agents, none of them support teams of agents in the specific sense

[5]The questionnaire can be found at *http://yallara.cs.rmit.edu.au/~kdam/Questionnaire/Questionnaire.html*

[6]Although we should note that the handling of protocols in Prometheus has been developed since the time of the questionnaire.

Concepts & Properties	**MaSE**	**Prometheus**	**Tropos**
Autonomy	H/M/DK	H/NA/H	H/M/M
Mental attitudes	L/M/H	H/M/H	H
Proactive	H/M/H	H/M/DK	H
Reactive	M	H/M/DK	H/L/DK
Concurrency	H/M/H	M/L/DK	H/M/H
Teamwork	H/M/H	N/L/NA	H/H/M
Protocols	H	M/H/M	NA/M/M
Situated	M/L/H	H	H
Clear concepts	SA/A/A	A/A/DA	SA/A/N
Concepts overloaded	A/N/SA	N	SDA/N/DA
Agent-oriented	SA/A/A	SA	SA/A/SA
Modelling & Notation			
Static+Dynamic	SA/A/A	SA/A/A	N/A/A
Syntax defined	A/A/SA	SA/A/A	SA/N/A
Semantics defined	A/SA/SA	A	SA/A/A
Clear notation	A	SA/A/A	SA/A/N
Easy to use	SA/A/A	A/N/A	SA/A/N
Easy to learn	N/N/A	SA/NA/SA	SA/N/A
Different views	N/N/A	A/A/SA	SA/A/N
Language adequate & expressive	SA/N/N	A	SA/A/N
Traceability	A/SA/SA	A	A/N/A
Consistency check	SA/A/SA	SA/A/A	_/A/DA
Refinement	SA/A/A	SA	SA/A/DA
Modularity	SA/A/A	SA/SA/A	SA/A/N
Reuse	N/SA/A	N/A/N	_/A/DA
Hierarchical modelling	N/A/A	SA/A/A	SA/A/DA
Process			
Requirements	SPEH	SPEH	SPE
Architectural design	SPEH	SPEH	SPE
Detailed design	SPEH	SPEH	SPE
Implementation	SEH/SPE/S	SPEH/S/n	SE/SPE/SPEH
Testing & Debugging	SPE/n/n	SPEH/S/n	n
Deployment	SE/SPE/SPEH	n	n
Maintenance	n/SPE/n	n	n
Pragmatics			
Quality	N/DA/A	A/N/N	DA/A/_
Cost estimation	_/DA/SA	DA/DA/N	DA/N/_
Management decision	_/DA/SA	SDA/N/_	SA/A/_
apps	21+	6-20	1-5
Real apps	no	no	no
Used by non-creators	yes	yes	yes/no/no
Domain specific	no	no	yes/no/no
Scalable	_/N/N	N/A/N	N/N/_
Distributed	_/SA/SA	SA/A/N	N/A/_

Figure 4: Comparing methodology's properties, attributes, process and pragmatics. Notation: L for Low, M for medium, H for High, DK for Don't Know, SDA for Strongly Disagree, DA for Disagree, NA for Not Applicable, N for Neutral, A for Agree, SA for Strongly Agree, _ for no response. S for Stage mentioned, P for Process given, E for Examples given, H for Heuristics given, n for none. The first two entries in each column are the developers of the methodology, the third is the student. A single entry in the column indicates that all three answers agreed.

of [6]. Both MaSE and Prometheus model the dynamic aspects of the system and handle protocols well. Tropos does not provide strong support for protocols, or for modelling the dynamics of the system except for some support at the detailed design level. According to the questionnaire, the concepts used in the methodologies tend to be clearly explained and understandable. However, the student who used Prometheus responded that there was some confusing terminology such as the distinction between percept, incident, trigger, and event. All three methodologies were perceived as being clearly agent-oriented.

Modelling Language: Overall, the responders felt that the methodologies' notations were clear and reasonably well defined (syntax/semantics) and fairly easy to use. Tropos was an interesting case: there was disagreement on whether the concepts were clear, and whether the notation was clear and easy to use; furthermore, there was disagreement on whether the syntax was defined, but oddly, there was consensus that the semantics were defined. Although one respondent felt strongly that MaSE's notation was adequate and expressive, the other two respondents disagreed (neutral). MaSE also does not claim to support different views (neutral from both creators). To some extent, the modelling language of all the methodologies supports traceability, i.e. the ability to track dependencies between different models. In terms of consistency checking, the level of support differs between methodologies. MaSE and Prometheus support it well whereas Tropos does not appear to support it. Refinement, modularity, and hierarchical modelling are generally well-supported (although there was disagreement from the student using Tropos) however reuse is not well handled by any of the methodologies.

Overall, the students had a very good impression of the notation of all the methodologies. For instance, Prometheus was highly appreciated, and the system overview diagram in particular was found to be useful. The students also reported some minor issues. For example, the Capability Diagram in Tropos is hard to draw since it is presented differently in [1] and [11]. There are some cases where the amount of text on arcs in the MaSE's concurrent diagrams makes them hard to read.

Process: From the software development life-cycle point of view, all of the methodologies cover the requirements, architectural design and detailed design. The students' responses to these phases of the three methodologies are also positive. They all said that the analysis stage of the methodology they had used was well described and provided useful examples with heuristics. This helped them to shift from object-oriented thinking to agent-oriented. The implementation phase is, surprisingly, not well supported; for example Tropos briefly explains that the concepts developed during design map to JACK constructs but does not provide a detailed process, heuristics, examples, or a discussion of the issues. Only MaSE and Prometheus mention testing/debugging. It is unclear to what extent MaSE supports it, while Prometheus' support is part of a research project [20] not yet integrated into tools for use by developers. Only MaSE discusses deployment and the level of support is unclear. Only one respondent (for MaSE) indicated any support for maintenance.

Pragmatics: As we have mentioned earlier, the pragmatics of a methodology plays a very important role in determining its applicability in industry as well as in academia.

In the questionnaire we asked the authors who the intended audiences for the methodology are. MaSE and Prometheus target undergraduate and industry programmers, whereas Tropos is aimed at experts. Furthermore, to measure how complex a methodology is to users, we used UML (Unified Modelling Language) and RUP (Rational Unified Process) as a benchmark. However, it is not clear that there is a consensus on the perceived complexity of UML+RUP, and so the answers to this question didn't allow any strong conclusions to be drawn. Regarding the availability of resources supporting the methodologies, most of them are in the form of conference papers, and journal papers or tutorial notes. None of the methodologies are published as text books. None of the methodologies seem to address issues such as quality assurance, or cost estimating guidelines. Although one respondent indicated that Tropos provides some support for decision making by management, e.g. when to move between phases, we do not agree with this assessment.

The availability of tool support also varies. MaSE and Prometheus are well supported with agentTool (MaSE) and JDE and PDT (Prometheus). Despite some minor issues, the use of agentTool really helped the student in drawing diagrams, checking model consistency and especially semi-automatically transforming analysis models to design constructs. PDT was also described by the student using it as being quite useful. Tropos has only weak tool support (a diagram editor). Although we attempted to determine how much "real" use (as opposed to student projects, demonstrators etc.) had been made of each methodology, it was not clear from the responses to what extent each methodology had been used, who had used the methodology, and what it had been used for.

5. CONCLUSION

We presented a comparison of three prominent methodologies. Overall, all three methodologies provide a reasonable support for basic agent-oriented concepts such as autonomy, mental attitudes, pro-activeness, reactiveness, etc. They all are also regarded by their developers and the students as clearly agent-oriented. In addition, the notation of the three methodologies is generally good. Regarding the process, all the methodologies provide examples and heuristics to assist developers from requirements gathering to detailed design. Implementation was supported to some degree by all methodologies whereas testing/debugging and maintenance are not clearly well-supported by any methodology. Additionally, some important software engineering issues such as quality assurance, estimating guidelines, and supporting management decisions are not supported by any of the methodologies.

5.1 Related Work

There has not been much work in comparing agent-oriented methodologies. Onn Shehory and Arnon Sturm [23] performed a feature-based evaluation of several AOSE methodologies. Their criteria included software engineering related criteria and criteria relating to agent concepts. In another paper [24] they used the same techniques in addition to a small experimental evaluation to perform an evaluation of their own Agent Oriented Modelling Techniques (AOMT). This work suffers from subjectivity in that the criteria they identified are those that they see as important and, naturally, AOMT focuses on addressing these criteria.

A framework to carry out an evaluation of agent-oriented analysis and design modelling methods has been proposed by Cernuzzi and Rossi [5]. The proposal makes use of feature-based evaluation techniques but metrics and quantitative evaluations are also introduced. The significance of the framework is the construction of an attribute tree, where each node of the tree represents a software engineering criterion or a characteristic of agent-based system. Each attribute is assigned with a score and the score of attributes on the node is calculated based on those of their children. They have applied that framework to evaluate and compare two AOSE methodologies: the Agent Modelling Techniques for Systems of BDI (Belief, Desire and Intention) Agents and MAS-CommonKADS.

In [15] O'Malley and DeLoach propose a number of criteria for evaluating methodologies with a view to allowing organisations to decide whether to adopt AOSE methodologies or use existing OO methodologies. Although they performed a survey to validate their criteria, they do not provide detailed guidelines or a method for assessing methodologies against their criteria. Their example comparison (between MaSE and Booch) gives ratings against the criteria without justifying them. Their work is useful in that it provides a systematic method of taking a set of criteria, weightings for these criteria (determined on a case by case basis), and an assessment of a number of methodologies and determining an overall ranking and an indication of which criteria are critical to the result.

5.2 Further Work

So far, the three methodologies have been compared based on the set of properties or attributes regarding concepts, notation, process and pragmatics. In addition to this, we intend to perform a *structural* comparison of the methodologies. This will involve examining their processes and models in detail and looking at commonalities and differences. For example, capturing goals and use cases is a feature of the requirements phase of a number of methodologies and this suggests that they are a useful activity. These and other issues if evaluated in detail may contribute another step towards developing the "next generation" of agent-oriented methodologies.

Acknowledgements

We would like to thank Hiten Bharatbhai Ravani, Robert Tanaman, Sindawati Hoetomo, Sheilina Geerdharry, and Yenty Frily for their work over the summer with the different methodologies. We would like to thank Lin Padgham for comments on a draft of this paper. We would also like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (ARC) under grant CO0106934.

6. REFERENCES

[1] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Troops: An agent-oriented software development methodology. Technical Report DIT-02-0015, University of Trento, Department of Information and Communication Technology, 2002.

[2] Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical

report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998.

[3] G. Caire, F. Leal, P. Chainho, R. Evans, F.G. Jorge, G. Juan Pavon, P. Kearney, J. Stark, and P Massonet. Project p907, deliverable 3: Methodology for agent-oriented software neginnering. Technical Information Final version, European Institute for Research and Strategic Studies in Telecommunications (EURESCOM), 09 2001.

[4] Giovanni Caire, Francisco Leal, Paulo Chainho, Richard Evans, Francisco Garijo, Jorge Gomez, Juan Pavon, Paul Kearney, Jamie Stark, and Philippe Massonet. Agent oriented analysis using MESSAGE/UML. In Michael Wooldridge, Paolo Ciancarini, and Gerhard Weiss, editors, *Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, pages 101–108, 2001.

[5] L. Cernuzzi and G. Rossi. On the evaluation of agent oriented modeling methods. In *Proceedings of Agent Oriented Methodology Workshop*, Seattle, November 2002.

[6] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.

[7] Scott A. DeLoach. Analysis and design using MaSE and agentTool. In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, 2001.

[8] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[9] Berard E.V. A comparison of object-oriented methodologies. Technical report, Object Agency Inc., 1995.

[10] U. Frank. Evaluating modelling languages: relevant issues, epistemological challenges and a preliminary research framework. Technical Report 15, Arbetsberichte des Instituts fuer Wirtshaftsinformatik (Universitt Koblenz-Landau), 1998.

[11] Fausto Giunchiglia, John Mylopoulos, and Anna Perini. The Tropos software development methodology: Processes, Models and Diagrams. In *Third International Workshop on Agent-Oriented Software Engineering*, July 2002.

[12] S. Hong, G. Van den Goor, and S. Brinkkemper. A formal approach to the comparison of object-oriented analysis and design methodologies. In *The Twenty-Sixth Annual Hawaii International Conference on System Sciences*, pages 689–699, Hawaii, 1993.

[13] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[14] Michael Luck, Peter McBurney, and Chris Preist. Agent technology: Enabling next generation computing: A roadmap for agent-based computing. AgentLink report, available from *www.agentlink.org/roadmap.*, 2003.

[15] S. A. O'Malley and S. A. DeLoach. Determining when to use an agent-oriented software engineering. In *Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001)*, pages 188–205, Montreal, May 2001.

[16] Lin Padgham and Michael Winikoff. Prometheus: A methodology for developing intelligent agents. In *Third International Workshop on Agent-Oriented Software Engineering*, July 2002.

[17] Lin Padgham and Michael Winikoff. Prometheus: A pragmatic methodology for engineering intelligent agents. In *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, pages 97–108, Seattle, November 2002.

[18] Lin Padgham and Michael Winikoff. Prometheus: Engineering intelligent agents. Tutorial notes, available from the authors, October 2002.

[19] Lin Padgham and Michael Winikoff. Prometheus: A brief summary. Technical note, available from the authors, January 2003.

[20] David Poutakidis, Lin Padgham, and Michael Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'02)*, 2002.

[21] Michael Prasse. Evaluation of object-oriented modelling languages: A comparison between OML and UML. In Martin Schader and Axel Korthaus, editors, *The Unified Modeling Language – Technical Aspects and Applications*, pages 58–75. Physica-Verlag, Heidelberg, 1998.

[22] J. Rumbaugh. Notation notes: Principles for choosing notation. *Journal of Object-Oriented Programming (JOOP)*, 8(10):11–14, May 1996.

[23] Onn Shehory and Arnon Sturm. Evaluation of modeling techniques for agent-based systems. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 624–631. ACM Press, May 2001.

[24] A. Sturm and O. Shehory. Towards industrially applicable modeling technique for agent-based systems (poster). In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, July 2002.

[25] B. Wood, R. Pethia, L.R. Gold, and R Firth. A guide to the assessment of software development methods. Technical Report 88-TR-8, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[26] M. Wooldridge, N.R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the third international conference on Autonomous Agents (Agents-99)*, Seattle, WA, May 1999. ACM.

[27] M. Wooldridge, N.R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3), 2000.

[28] E. Yu. *Modelling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto, Department of Computer Science, 1995.