USING A TWO-WAY BALANCED INCOMPLETE BLOCK DESIGN TO COMPARING AN AGENT-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES

Faezeh Parandoosh and Siavosh Kaviani
Department of Computer Science, Payamenour University, Tehran, Iran

Keywords: Incomplete block design, Agent, Agent-Oriented Software Engineering (AOSE), Agent-Oriented methodologies, MaSE, Prometheus, Tropos and Gaia.

Abstract: There has been a surge of interest in agent-oriented software engineering in recent years. Numerous methodologies for developing agent-based systems have been proposed in the literature and the area of agent-oriented methodologies is maturing rapidly. Evaluating methodologies' strengths, weaknesses and domains of applicability plays an important role in improving them and in developing the "next-generation" of methodologies. In this paper, we present a reliable framework that adopts statistical techniques to compare agent-oriented methodologies. Based upon this framework we performed a comparison of four AOSE methodologies MaSE, Prometheus, Tropos and Gaia.

1 INTRODUCTION

Agent-oriented techniques represent an exciting new means of analyzing, designing and building complex software systems. They have the potential to significantly improve current practice in software engineering and to extend the range of applications that can feasibly be tackled.

"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." (Luck, McBurney & Preist, 2003, p.11).

Even though AOSE methodologies have been proposed, few are mature or described in sufficient detail to be of real use. In fact, 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 (Castro, Kolp & Mylopoulos, 2002; Bersciani et al. n.d.).

An important step is to understand the differences between the various key methodologies, and to understand each methodology's strengths, weaknesses, and domains of applicability.

In this paper we perform the comparison on several well-known methodologies. A qualified methodology has been selected based on the availability of the documentation that describes it, the familiarity of the agent community with it, and its domain of applicability. As a result, the following 9 methodologies have been selected as treatments to our experiment: Gaia, MaSE, Prometheus, Tropos, MAS-CommonKADS, MESSAGE, FIPA-OS, JiVE and CNFM (Elamy and Far, 2005). In this paper we perform a comparison on four well-known AOSE methodologies MaSE, Prometheus, Tropos and Gaia.

In section 2, we briefly introduce these methodologies. In section 3, we describe a framework for comparing AOSE methodologies. We then (section 5) select the Appropriate Statistical Model techniques to compare AOSE methodologies. In section 6 we apply the framework to compare the methodologies.

2 THE METHODOLOGIES

2.1 MaSE

Multi agent Systems Engineering (MaSE) (Scott et al.2001) is an agent-oriented software engineering methodology which is an extension of the object-oriented approach. As a software engineering

Parandoosh F. and Kaviani S. (2007).
USING A TWO-WAY BALANCED INCOMPLETE BLOCK DESIGN TO COMPARING AN AGENT-ORIENTED SOFTWARE ENGINEERING METHODOLOGIES.
In *Proceedings of the Second International Conference on Evaluation of Novel Approaches to Software Engineering* , pages 56-65
DOI: 10.5220/0002585200560065

methodology, the main goal of MaSE 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*. This goal model is a product of a goal decomposition process in that goals are broken down in subgoals, subgoals to sub-subgoals and so on. The second step involves *Use Cases*, a technique which is commonly found in object-oriented methodologies. It includes extracting main scenarios from the initial system context or copying them from it if they exist. The use cases should show how a goal can be achieved during a normal system operation as well as erroneous conditions. The second part of this step is to apply those use case. Firstly, an initial set of roles is identified based on goals and use cases scenarios. Secondly, the sequence of events that occur in the interaction or communication between roles is represented in a *Sequence Diagram*. This model is also analogous to UML sequence diagrams except that entities are roles rather than objects. *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. 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 agent Tool. Its latest version 2.0 implements all seven steps of MaSE.

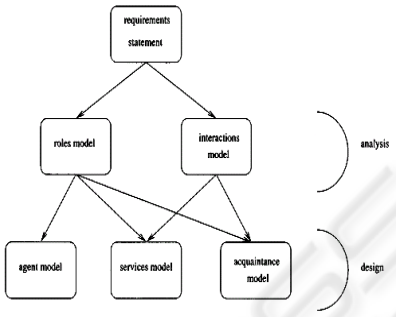

Figure 1: Relationship between Gaia's models.

2.2 Gaia

Gaia is one of the first methodologies which is specifically tailored to the analysis and design of agent-based systems. Its main purpose is to provide the designers with a modeling framework and several associated techniques to design agent-oriented systems.

The Gaia methodology is both general, in that it is applicable to a wide range of multi-agent systems, and comprehensive, in that it deals with both the macro-level (societal) and the micro-level (agent) aspects of systems. Gaia separates the process of designing software into two different stages: *analysis* and *design* (Wooldridge, Jennings & Kinny, 1999; Wooldridge, Jennings & Kinny, 2000).

Analysis involves building the conceptual models of the target system, whereas the design stage transforms those abstract constructs to concrete entities which have direct mapping to implementation code. Figure 1 depicts the main artifacts of each stage: *Role Model* and *Interaction Model* (Analysis), and *Agent Model, Services Model*, and *Acquaintance Model* (Design).

2.3 Prometheus

The Prometheus 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 (Giorgini & Henderson-Sellers, 2005; Padgham & Winikoff, 2002).

The **system specification** is the first phase of Prometheus. Its main purpose is building the *system's environment model*, *identifying the goals* and *functionalities of the system*, and *describing key use case scenarios*.

The **architectural design** is the second phase of Prometheus. The three main activities involved in this stage are: *defining agent types*, *designing the overall system structure*, and *defining the interaction between agents*.

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. (Figure 2)

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.

2.4 Tropos

Tropos is an agent-oriented software development methodology created by a group of authors from various universities in Canada and Italy (Bersciani et al. 2004; Fatemi, NematBakhsh & TorkLadani, 2006; Giunchiglia, Mylopoulos & Perini, 2002; Leskowsky & Anderson, 2003). Tropos is based on two key ideas. First, the notion of agent and all related mentalistic notions (for instance goals and plans) are used in all phases of software development, from early analysis down to the actual implementation. Second, Tropos covers also the very early phases of requirements analysis, thus allowing for a deeper understanding of the environment where the software must operate, and of the kind of interactions that should occur between software and human agents. The Tropos language for conceptual modeling is formalized in a metamodel described with a set of UML class diagrams.

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 analyzed. 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*.

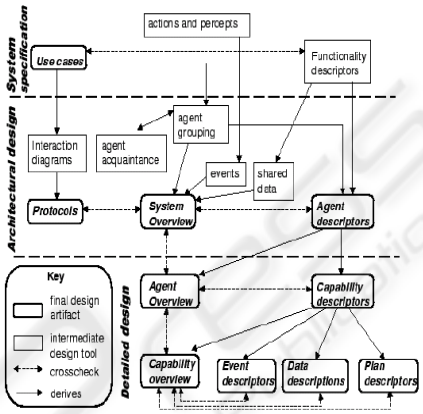

Figure 2: Overview of 3 Phases in Prometheus.

3 A COMPARISON FRAMEWORK

In this section, we briefly describe a methodology evaluation framework within which the methodology comparison is conducted.

The comparison framework covers nine major aspects of each AOSE methodology: **Concepts, Upgradeability**, **Modeling language**, **Basic properties**, **Mental attitudes**, **Modeling**, **Process, Pragmatics and Management.** This framework is adapted from a framework proposed in (Fatemi, NematBakhsh & TorkLadani, 2006; Giorgini & Henderson-Sellers, 2005) for comparing Agent-Oriented Methodologies.

3.1 Concepts

Agent-oriented concepts are of great importance for agent-oriented methodologies in general and for agent-oriented modeling languages in particular. *Usability*- to what extend the methodology is easy to use and implement, *Modeling*- to what extend concepts can be expressed in multiple models/diagrams; *Overloading*- to what extend concepts can be overloaded; *Notation*- to what extend notations are semantically and syntactically clear and simple across models?

3.2 Upgradeability

Mobility- how capable is the methodology in modeling agent migration? *Scalability*- This measures 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. *Open Systems Support*- to what extend it can provide support for open systems to allow dynamic integration/removal of new agents/resources; *Distribution*- This criterion measures the methodology's support for designing distributed systems. It means the methodology should provide mechanisms, including techniques and models, to describe the configuration of processing elements and the connection between them in the running system. It shows not only the physical of the different hardware components that compose a system, but also the distribution of executable programs on this hardware. More specifically, such models need to depict the deployment of agents over the network. *Dynamic Structure*- to what extend it can provide support for dynamic system reconfiguration when agents are created or destroyed.

3.3 Modeling Language

Clarity and understandability- These two criteria are closely related to each other and both of them are fundamental requirements of a modeling language. In fact, a methodology which provides clear notations tends to increase the users' understandability of the models. *Consistency*- to what extend it satisfies modeling consistency (i.e. no individual requirement is in conflict); *Unambiguity*- symbols and syntax are provided to users so that they can build a representation of a particular concept. Thus, the semantic or meaning of a concept is the users' interpretation of the representation provided. However, this interpretation can be different from observer to observer, which in turn results in misunderstandings. Therefore, it is important to make sure that a constructed model can be interpreted unambiguously. *Traceability*- There are relationships between models and between models and the requirements of the target system. Traceability requires that it has to be easy for the designers and the audiences of the design documents to understand and trace through the models. This may increase the users' understanding of the system. Tracing backwards and forwards between models and stages also allow the users to verify that all the requirements of the system are addressed during the analysis and design stages. Traceability also assists the designer produce new models by referring to the models that have been previously constructed. A result of doing this may be increased productivity in the sense that information gathered from one model can be used to construct others. *Usability*-It is important for a modeling language not only to be understandable to the users but also to be easy to use. The first step toward using a modeling language is to learn the notation. Hence, it is desirable that the notation be easy to learn by both expert and novice users. In addition, the easier the users can remember the notation, the quicker they are able to learn to use it. Therefore, the notation should be as simple as possible. Furthermore, since people usually sketch models by hand during the process of brainstorming or reviewing designs, it is essential for the notation be easy to draw and write by hand. Finally, as mentioned earlier, one of the important purposes of a modeling language is to convey information among the users. Often this is in the form of hardcopy documentation for reading and discussing. Hence, it is important that the diagrams produced are easy to read and comprehend when printed.

3.4 Basic Properties (Weak Notation)

Autonomy- Agents can operate and make their own decision on which action they should take without direct intervention of humans or other agents. In other words, both agents' internal state and their own behaviour are controlled by themselves. We want to know does the methodology support modeling a decision-making mechanism of agents regardless of the environment or does the methodology support describing an agent's self-control features? For example, functionalities and tasks being encapsulated within an agent may increase the degree of autonomy. *Reactivity*/*Proactivity*- Pro-activeness is an agent's ability to pursue goals over time and Reactivity an agent's ability to respond in a timely manner to changes in the environment, so the degree of allowing reactivity/proactivity is very important; *Sociability*- to what degree the methodology provides organized relationships among agents, represents agents' commitments and interfaces with other entities; *Adaptability*- to what degree agents are flexible to adjust their activities to environmental changes; *Concurrency*- agent's ability to deal with multiple goals and/or events at the same time. More specifically, agents are able to perform actions/tasks or interact with other agents simultaneously. We want to know what extend

agents performs several tasks simultaneously; *Interactions*- to what extend agents can interact with other agents, and with environment; *Human Computer Interaction* to what degree the methodology is capable to construct models to represent user interfaces and system-user interaction?

3.5 Mental Attitudes (Strong Notation)

This feature relates to the strong agency definition of agents. The three major elements of the Belief-Desire-Intention (BDI) architecture of agents are an example of it. The BDI architecture defines an agent's internal architecture by its beliefs, the desires or goals it wants to achieve and its intentions or plans to accomplish those goals. We want to know does the methodology support modeling mental attitudes of agents.

3.6 Modeling

Agent-Oriented- how efficient the methodology is in supporting modularity, hierarchical modeling, reusability, and traceability; *Abstraction*- to what degree it is efficient to produce models at various levels of details and abstractions; *Consistency*- Models should not contradict each other. This property becomes more important as the design evolves. More specifically, the representation of various aspects of a system such as structure, function and behaviour should be consistent. We want to know does the methodology provide guidelines and techniques for consistency checking both within and between models, do the methodology supported by tools that provide model consistency checking or is data dictionary used to avoid naming clashes between entities.

3.7 Process

An ideal methodology should cover six stages, which are enterprise modeling, 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. *Development principles*- This criterion addresses the lifecycle coverage in a broad view. It examines the development stages and their corresponding deliverables described within the methodology. In addition, it examines the supporting software engineering lifecycle model and development perspectives. *Process step*- Differing to the above criterion, this one measures the lifecycle coverage in more detail. In fact, an important aspect in evaluating whether a methodology covers a particular process step is the degree of detail provided. *Estimating and quality assurance guidelines*: These two criteria determine if such guidelines are provided within the methodology process. Estimating guidelines are important to task planning. Quality assurance guidelines provide the assessors with useful information in evaluating the merit of the delivered product. *Supporting development context*: This criterion identifies the development context supported by the methodology. A development context specifies a set of constraints within which the software development has to take place.

3.8 Pragmatics

Maturity: The maturity of a methodology is a factor that can play an important role in determining the quality of a methodology. There are several ways to measure the maturity of a methodology, for example: What are available resources supporting the methodology? What is the methodology's "experience" such as the history of the methodology use? What are available resources supporting the methodology? *Domain applicability*: This considers whether the methodology is targeted at a specific type of software domain such as information systems, real time systems or component based systems. With regard to this issue, the methodology that is applicable to a wide range of software domains tends to be more preferred.

3.9 Management

Management Decision- to what degree the methodology can be accepted by management; *Cost Estimation* - to what degree it is economically feasible?

4 SELECTING PARTICIPANTS

There were 8 participants in the experiment (evaluators), all of whom were taking a specialized graduate course in Agent-Oriented Software Engineering. They all had adequate knowledge and experience in software development. The participants were provided sufficient documentation about the methodologies, clear instructions about the experiment, and equal amount of time to complete

their task using well-prepared surveying questionnaires.

5 SELECTING THE APPROPRIATE STATISTICAL MODEL

In (Elamy and Far, 2005), they compared 9 methodologies and the statistical model was One-way ANOVA model for a Complete Random Design (CRD), they decided to have at least 4 replicas for each of the 9 treatments. they have two limitations there. First, they suspect that there is heterogeneity among evaluators for many reasons, such as technical experience; although the randomization will tend to spread the heterogeneity around to reduce bias, we still have a strong limitation, which is the lack of resources; they have only 12 participants. To bypass this lack, they decided to make use of each participant to assess more than one methodology. This solution headed us to consider a 2-way ANOVA model with blocking.

In this model, each block of treatments will be assigned at random to one participant. If we used a Randomized Complete Block Design (RCBD), each evaluator should assess a complete block, i.e. 4 methodologies, and the design will probably more effective because we will have more replications, $8 \times 4 = 32$, by assuming considerable variability within blocks. Unfortunately, this design has also a limitation that makes it hard to implement, because not all the participants are familiar with the whole methodologies.

After a thorough analysis and extended discussions, they ended up the battle by adopting the two-way Balanced Incomplete Block Design (BIBD) *additive* model with *fixed*-effects,

$$Y_{ij} = \mu + \alpha_i + \beta_j + \in_{ij}$$

Where

i =1 to t; t = number of treatments = 4

j=1 to b; b = number of blocks = 8

Y_{ij} =the observation (effectiveness) recorded on the treatment (methodology) i by the evaluator assigned to the block j

μ = overall mean of experiments of this type

α_i =treatment main effects, the deviation from the mean caused by the i_{th} treatment; $\sum \alpha_i = 0$ (fixed effects)

β_i =main effects of the column blocking variable, the deviation from the mean caused by the j_{th} block; $\sum \beta_i = 0$ (fixed effects)

$\in_{ij}$ =independent random error $\sim N(0,\sigma^2)$

and

k = number of treatments per block = 3; r = number of replications per treatment = 4; λ = number of times any pair of treatments appears together in the same block = $r(k-1)/(t-1) = 1$

This model is referred to as "**balanced**" because each block will have the same number of treatments, "**additive**" because no interactions are considered between factors, "**incomplete**" because each participant will not evaluate a complete set of treatments (4 methodologies), "**fixed**" because we were limited to narrow down the selection of the qualified methodologies upon our own interest by looking into specific measures, and not at random from a large number of methodologies; thus, if we decided to repeat the experiment we would use the same methodologies, and not reselecting new ones randomly. In fact, this model is applicable for both RCBD and BIBD; however, the analysis is different. By denoting the nine methodologies with letters from A to D as shown in Table 1, we can obtain 16 replicas; which satisfies our goal of having 4 replicas for each treatment.

Table 1: BIBD assignment.

Block (evaluators), k	Treatments(Methodologies),i				Y_i
	M1	M2	M3	M4	
E1	A	B			$Y_{.1}$
E2	A	B			$Y_{.2}$
E3			C	D	$Y_{.3}$
E4			C	D	$Y_{.4}$
E5	A	B			$Y_{.5}$
E6	A	B			$Y_{.6}$
E7			C	D	$Y_{.7}$
E8			C	D	$Y_{.2}$
$Y_{j.}$	$Y_{1.}$	$Y_{2.}$	$Y_{3.}$	$Y_{4.}$	$Y_{..}$

M=Methodology; E=Evaluator

6 COMPARING METHODOLOGIES

Based on the comparison framework (Section 3), Statistical Model proposed in section 5 and based on the users view points; we have performed a brief comparison between selected methodologies that is explained below.

6.1 Concepts

Usability-We attempted to measure how complex a methodology is to users, by using UML (Unified

Modeling Language) and RUP (Rational Unified Process) as a benchmark. There seems to be an agreement among the respondents that the four methodologies are about the same complexity as UML and RUP. However, it is not clear that there was a consensus on the perceived complexity of UML+RUP, and so the answers to this question did not allow any strong conclusions to be drawn. *Notation*- the responders generally agreed that the methodologies' notation were clear and the symbols and syntax are well defined. These indicate the notations provided by all of the four methodologies are fairly clear and understandable.

6.2 Upgradeability

Dynamic Structure and Scalability- Regarding this criterion, most of the respondents stood on a neutral point of view. In our perspective, this issue is not explicitly addressed in any of the methodologies. More specifically, they do not tell how to deal with the introduction of new components or modules in an existing system. *Open Systems Support*- none of the methodologies support design of open systems. *Distribution*- Overall, all of the methodologies implicitly supports distribution. This is partially due to the nature of agent-based systems. When developed, agents communicate with each other via a message passing system. In other words, agents are not coupled until an interaction needs to occur. As a result, the agents do not necessarily populate on the same systems. The results from the questionnaire also agreed with that view. Responses on this criterion on average range from Neutral to Agree. MaSE is an exceptional case in that the system design step of MaSE allows the developers to design and allocate agents over the network. It is supported by the Deployment Model, a representation of agent types and their location on the network. Therefore, we tend to strongly agree with all the respondents of MaSE that the methodology provides sufficient support for distribution.

6.3 Modeling Language

Clarity and understand ability- the responders generally agreed that the methodologies' notation were clear and the symbols and syntax are well defined. These indicate the notations provided by all of the four methodologies are fairly clear and understandable. *Consistency*- In terms of consistency checking, the level of support differs between methodologies. MaSE and Prometheus support it well whereas Tropos and Gaia do not appear to support it. These responses seem to relate the availability of tool support integrated with the methodology. PDT (Prometheus) and agent Tool (MaSE) provides a strong support for model and design consistency checking. *Traceability*- Likewise to consistency, MaSE and Prometheus appear to be the leader in terms of supporting this feature. The responders of these two methodologies, including us agreed that there are clear links between models provided by them. For instance, goals, roles, agents, and tasks are all linked together. This strong connection improves the ability to track dependencies between different models. Such connections, as described in one of the paper related to MaSE, allow developers to (automatically or manually) derive design models (e.g. an agent's internal architecture) from analysis constructs. *Unambiguity*- Semantics is also well-defined by all the methodologies. For Gaia, the student felt that the semantics of Gaia's modeling language are not well-defined. However, we tend to disagree with that. The meaning of symbols and models in Gaia is in fact defined in detail. Tropos was another 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. For the remaining two methodologies, there was an agreement (strongly agree - agree) that the modeling language is unambiguous in the sense that the semantics of the notation is clearly defined. *Usability*- Overall, most of the respondents agreed that the notations of the four methodologies are easy to learn and use. This also relates to the agreement of the understand ability and clarity of the notation as discussed above. Tropos is, however, an exception case. One of its authors strongly agrees and the other agrees that the modeling language of the methodology is easy to use. In contrast, the user and we preferred to take a neutral view on this criterion. This is due to the fact, unlike MaSE and Prometheus, Tropos does not have a tool support integrated with the methodology. As a result, users may find it difficult to draw diagrams, checking model consistency, etc.

6.4 Basic Properties (Weak Notation)

Autonomy- According to the responses from the survey and our assessment, all of the four agent-oriented methodologies recognize that importance. The level of support for autonomy in all of them is overall "good" (ranging from medium to high). This

is reacted by the fact that all the four methodologies provide various supports for describing an agent's self-control features. For instance, functionalities and tasks are encapsulated within an agent. In addition, plan diagrams in Tropos, concurrent task diagrams in MaSE, or plan descriptors in Prometheus allow the decision-making mechanism of agents to be modelled regardless of the environment and other entities. That mechanism is based upon the agents' goals and their roles within the system. *Pro-activeness and reactivity*- Based on the results, it seems that these two attributes are difficult to measure we received highly varying responses. They seem to be fairly well supported by some of the four methodologies (medium-high for MaSE and Prometheus, mostly high for Tropos). Similarly to mental attitudes, this can be explained by the fact that in these three methodologies agents' goals are captured and so are the execution of plans (i.e. actions or tasks) to achieve these goals. In addition, Prometheus has the action descriptors which are a means for specifying agents' responses to environment changes in terms of external events. *Concurrency*- In terms of support for concurrency, although the ratings are mostly medium high and vary considerably, MaSE is probably best with its concurrent task diagrams and communication class diagrams. The former is used to specify how a single role can concurrently execute tasks that define its behaviour. The latter, also expressed in the form or a finite state machine, is able to define a coordination protocol between two agents. Prometheus was rated as being one of the weakest although we should note that the handling of protocols in Prometheus has been developed since the time of the questionnaire. *Sociability*- Although the methodologies all support cooperating agents, none of them support teams of agents in the specific sense of teamwork. All of the methodologies provide a wide range of communication modes. More specifically, they support both direct/indirect and synchronous/asynchronous communication.

6.5 Mental Attitudes (Strong Notation)

Prometheus and Tropos support well (medium to high) the use of mental attitudes (such as beliefs, desires, intentions) in modeling agents' internals. The percept and action descriptors in Prometheus and the actor diagrams in Tropos represent the agent knowledge of the world (i.e. beliefs). Goals and plans are also modelled in the two methodologies. In contrast, MaSE and Gaia provide weaker support for capturing an agent's mental attitudes. MaSE have goal diagram but they do not have a representation of the agent's belief.

6.6 Modeling

Agent-Oriented- To some extent, all the methodologies supports traceability, In terms of consistency checking, the level of support differs between methodologies. MaSE and Prometheus support it well whereas Tropos and Gaia do not appear to support it. Modularity and hierarchical modeling are generally well-supported (although there was disagreement from the student using Tropos) however reusability is not well handled by any of the methodologies. *Consistency*- MaSE and Prometheus support it well whereas Tropos and Gaia do not appear to support it.

6.7 Process

Development principles- From the software development life-cycle point of view, all four methodologies cover requirements analysis, and architectural design. Some of them (MaSE and Prometheus) go further than that with description of detailed design, implementation and testing/debugging. Deployment is only addressed in MaSE. *Process steps*: The process steps described in the requirements analysis and design phases are also addressed well in most of the four methodologies. Detailed design is not well documented in Gaia. Furthermore, a common feature in all the methodologies is the lack of management making decisions in performing the process steps such as when to move to the next phase, etc. *Estimating and quality assurance guidelines*: Because of the immaturity of agent-oriented methodologies, issues relating to cost estimating or quality assurance are not addressed in all four methodologies. They probably rely on the current software engineering practice of these matters. *Supporting development context*: Top down design and "Greenfield" development is the popular approaches employed by most of the four methodologies.

6.8 Pragmatics

Maturity- Regarding the availability of resources supporting the methodologies, most of them are in the form of conference papers, journal papers or technical reports. The availability of tool support is varies. MaSE and Prometheus are well supported with agentTool (MaSE) and JDE and PDT (Prometheus). According to the authors of MaSE

(based on the questionnaire's responses), agentTool can be used as a diagram editor, a design consistency checker, code generator and automatic tester. They also revealed that agentTool has been downloaded and used by many people in academia as well as industry and government. The tools supporting Prometheus, PDT and JDE, also provide a similar range of functionalities. PDT supports for drawing diagrams, checking model and design consistency, and generating reports. JDE (Jack Development Environment) can be used a design tool to build the structure of an agent system in which the concepts provided by JACK match the artifacts constructed in Prometheus' detailed design phase. Tropos has only weak tool support (a diagram editor) whereas there is no tool support for Gaia that we are aware of. 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. Nevertheless, to our knowledge, MaSE was used to design a team of autonomous, heterogeneous search and rescue robots (Scott et al. 2002). Tropos was used to develop a web-based broker of cultural information and services for the government of Trentino, Italy (Bresciani et al. 2002) and an electronic system called Single Assessment Process to deliver an integrated assessment of health and social care needs for older people (Mouratidis et al. 2002). *Domain applicability*: The respondents tended to agree that there is no limitation to the application domains where one of the four agent-oriented methodologies can be applied.

6.9 Management

Cost Estimation- None of the methodologies seem to address cost estimating guidelines. *Management Decision*- None of the methodologies seem to address management decision, 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.

7 CONCLUSION AND FUTURE WORKS

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. 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. An important step towards a complete unique methodology is to understand the differences between the various key methodologies, and to understand each methodology's strengths, weaknesses, and domains of applicability.

In this paper, we use a new statistical approach, based on adopting the incomplete block design model to evaluate the four AOSE methodologies. Overall, all four methodologies provide a reasonable support for autonomy, mental attitudes, pro-activeness, and reactive ness. The notation of the four methodologies is generally good. Most of them have a strong modeling language in terms of satisfying various criteria such as clarity and understandability, adequacy and expressiveness, ease of use, and unambiguity. However, there are several exceptions. Tropos was not perceived as being easy to use whilst GAIA was both ranked weakly on adequacy and expressiveness. In addition, only Prometheus and MaSE provide techniques and tools for maintaining the consistency and traceability between models. For the other two methodologies, there is still more room for improvement with respect to these issues. It is also emphasized that none of the evaluated methodologies explicitly provide techniques, guidelines, or models to encourage the design of reusable components or the reuse of existing components. Regarding the process, only Prometheus and MaSE provide examples and heuristics to assist developers from requirements gathering to detailed design. Gaia was not support detailed design. Additionally, even though all phases from early requirements to implementation are mentioned in Tropos with examples given, the methodology does not appear to provide heuristics for any phase. Implementation, 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.

As agent-oriented methodologies continue to be developed, research will keep aiming at the direction of determining which agent-oriented methodologies are best suited to support the development of a particular project or system. Hence, there are various future works that can be done in this area.

ACKNOWLEDGEMENTS

We would like to thank Dr.Faraahi, for his valuable comments on a draft of this paper.

REFERENCES

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A. (2004) *Troops: An agent-oriented software development methodology*. From Trento University, Department of Information and Communication Technology.

Fatemi, A., NematBakhsh, N.,TorkLadani, B.(2006). An Investigation of Agent Oriented Software Engineering Methodologies to Provide an Extended Methodology, *IEEE International conference on Information and Communication Technologies*, 0-7803-9521.

Elamy, H., Far, H. (2005) Utilizing incomplete block designs in evaluating agent-oriented software engineering methodologies, *IEEE International conference on Electrical and Computer Engineering*, 0-7803-8886.

Giorgini, P., Henderson-Sellers, B. (2005). *Agent-Oriented Methodologies,* IDEA GROUP. United States of America, 1th edition.

Luck, M., McBurney, P., Preist, C. (2003). Agent technology: Enabling next generation computing: A roadmap for agent-based computing. AgentLink report. Retrieved from www.agentlink.org/roadmap.

Wooldridge, M., Jennings, N.R. & Kinny,D. (1999). A Methodology for Agent-Oriented Analysis and Design. In *Proceedings of the third international conference on Autonomous Agents and Multi-Agent Systems*, Seattle, WA.

Wooldridge M., Jennings N.R., Kinny, D. (2000) , *The Gaia Methodology for Agent-Oriented Analysis and Design*, From Liverpool University, Department of Computer Science.

Giunchiglia, F., Mylopoulos, J., Perini, A. (2002). The Tropos software development methodology: Processes, Models and Diagrams. In *Third International Workshop on Agent- Oriented Software Engineering*.

Wooldridge, M., (1997) Agent-based software engineering In *IEE Proc. on Software Engineering,* 144 (1) 26-37.

Waldner, A., Fuster , P., (2002). An Agent Oriented Methodology: High Level and Intermediate Models.

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

Padgham, L., Winikoff, M., (2002). Prometheus: A methodology for developing intelligent agents. *In Third International Workshop on Agent-Oriented Software Engineering*.

Leskowsky, Z. and D. Anderson (2003). Comparing Agent Oriented Methodologies. American Society of System.

Castro, J., Kolp, M., Mylopoulos, J., (2002). Towards requirements-driven information systems engineering: The tropos project, *Information Systems*, Elsevier: Amsterdam, The Netherlands.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos J., & Perini, A. Towards an Agent Oriented approach to Software Engineering", *In the Workshop Dagli oggetti agli.*

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J and Perini, A. (2002). *Troops: An agent-oriented software development methodology*. Technical Report DIT-02-0015, from Trento University, Department of Information and Communication Technology.

Scott A. DeLoach, Eric T. Matson, and Yonghua Li. (2002). Applying agent oriented software engineering to cooperative robotics. *In Proceedings of the The 15th International FLAIRS Conference*.

Mouratidis, H., Giorgini, P., Manson, G., & Philp, L., (2002). Using Tropos methodology to Model an Integrated Health Assessment System. *In Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems*.