A Framework for Evaluating Agent-Oriented Methodologies

Arnon Sturm
Technion - Israel Institute of Technology
Haifa 32000, Israel
sturm@techunix.technion.ac.il

Onn Shehory
2nd IBM Haifa Research Lab, Haifa University
Haifa 31905, Israel
onn@il.ibm.com

ABSTRACT
Multiple agent-oriented methodologies were introduced in recent years, however no systematic evaluation of these was offered. As a result, it is difficult to select a methodology for a specific project. Additionally, there are no means for determining what the advantages and drawbacks of each methodology are. To resolve these problems, we devise a framework for evaluating and comparing agent-oriented methodologies. This framework focuses on four major aspects of a methodology: concepts and properties, notations and modeling techniques, process, and pragmatics. We demonstrate the usage of the suggested framework by evaluating the GAIA methodology. As sought, this evaluation identifies the strengths and the weaknesses of GAIA, thus exemplifying the capabilities of our framework.

Categories and Subject Descriptors
D.2.1 [**Software Engineering – Requirements/Specifications**]: Methodologies.

D.2.8 [**Software Engineering - Metrics**]: Complexity measures, Process metrics.

General Terms
Management, Measurement, Documentation, Design.

Keywords
Agent-oriented software engineering, evaluation of methodologies, comparison of methodologies, agent-oriented methodologies

1. INTRODUCTION
During the last decade, many methodologies for developing agent-based systems have been developed. A methodology is the set of guidelines for covering the whole lifecycle of system development both technically and managerially. A methodology, according to [7], should provide the following: a full lifecycle process; a comprehensive set of concepts and models; a full set of techniques (rules, guidelines, heuristics); a fully delineated set of deliverables; a modeling language; a set of metrics; quality assurance; coding (and other) standards; reuse advice; and guidelines for project management. The relationships between these components are shown in Figure 1.

There are more than two dozens agent-oriented methodologies. The multiplicity and variety of methodologies result in the following problems: (1) Industrial problem: selecting a methodology for developing an agent-based system/application becomes a non-trivial task, in particular for industrial developers which hold specific requirements and constraints; (2) Standards problem: multiple different methodologies are counter-productive for arriving at a standard. With no standard available, potential industrial adopters of agent technology refrain from using it; (3) Research problems: excessive efforts are spent on developing agent-oriented methodologies, in times producing overlapping results. Additionally, as a result of allocating resources to multiple methodologies, no methodology is allocated sufficient research resources to enable addressing all aspects and providing a full-fledged agent-oriented methodology. In this paper we provide means for addressing these problems by supplying a framework for evaluating agent-oriented methodologies. This evaluation framework may be used by organizations to select a methodology for developing agent-based applications. It can also help researchers to examine the similarity and the differences among existing agent-oriented methodologies and to analyze the needed attributes of such methodologies. Additionally, setting a scale for grading agent-oriented methodologies, and using the scale in conjunction with our framework, may result in a selection of the better methodologies, gradually reducing their number. This selection may eventually converge to a small set of the most fit agent-oriented methodologies, possibly leading to standardization.

Agent-oriented methodologies can be classified into two major classes: general-purpose methodologies and domain-specific methodologies. The framework provided in this paper is aimed for the evaluation of general-purpose methodologies, e.g., GAIA [17],[19], Tropos [2], and MaSE [6]. Yet, evaluating these methodologies introduces several difficulties:

- Comparing methodologies is often difficult, because they might address different aspects or differ in their terminology. For example, they are based on different concepts – MaSE is based on software engineering concepts (such as state machines and components) and Tropos is based on knowledge level concepts (such as actors, goals, dependencies, and plans).
- Some of the methodologies are influenced by a specific approach, e.g., BDI or OO.
- The completeness of various methodologies varies dramatically. For example, some provide only the process, some present graphical notations, while others integrate several aspects of a methodology (i.e., process, notations, guidelines, etc.).

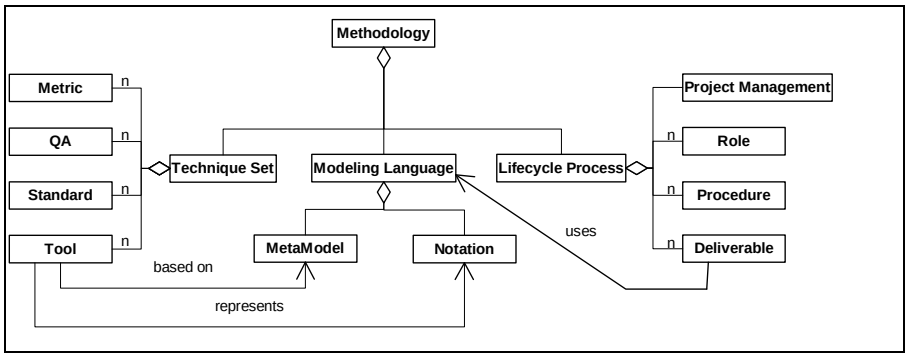

Figure 1. The components of a methodology and the relationships among them

Since the agent-oriented paradigm can be considered as an evolution of the object-oriented paradigm, we considered the type of evaluations made for the object-oriented methodologies. Such evaluations have been discussed by many studies as indicated by [15]. These evaluations and comparisons suffer from a number of common flaws as follows:

- Using an inappropriate framework for performing the evaluation.
- There is no agreement on what a methodology is and on what it should consist of.
- Sometimes there is no ranking for the support of a particular concept of a specific methodology. This leads to unsatisfactory results of the evaluation.
- In many cases the evaluation cannot be repeated.

A few evaluations of agent-oriented methodologies have been suggested. In [18], the authors set a list of questions that a methodology should address. However, neither evaluation nor a comparison has been performed using that set. Another study [4], suggests a framework for evaluating agent-oriented methodologies, however, the compared criteria refer only to the expressiveness of the methodologies and not the wider set encompassed within the methodology definition. In [9], the author performs an evaluation of five agent-oriented methodologies, however, he refers only to some supported concepts such as organization design and cooperation and not to the broad set of attributes that constitute a complete methodology. In [11], the authors perform an evaluation of only the modeling part within a methodology. Other studies that deal with evaluating agent-oriented methodologies compared two or three methodologies, yet mainly with respect to the expressiveness and the concepts supported by the methodology.

In this paper, we provide a comprehensive framework of evaluating and comparing agent-oriented methodologies. This framework offers a well-defined, structured set of aspects that an agent-oriented methodology should include. The provided framework is a qualitative one, however, it can be transformed into a quantitative one borrowing the concepts from [4]. The suggested framework, although based on the framework suggested by [15], extends and modifies it to address the unique requirements of agent-oriented methodologies. The framework can be used for various evaluation techniques:

- Feature analysis – The evaluation is done by referring to the available resources.
- Survey – The evaluation is done by examining the results of the survey that is distributed among practitioners and researchers. As a result of the size of the population surveyed, these results may be statistically justified.
- Case studies – The evaluation is done by examining the results of case studies.
- Field experiments – The evaluation is done by examining the results of field experiments.

Due to lack of space we will not discuss the advantages and drawbacks of each technique. However, such a discussion can be found in [12].

The specific use of these techniques is determined mainly by resource constraints. In this paper, we perform the evaluation using the feature analysis technique to demonstrate its applicability and ease of use. An evaluation of this type can be easily performed by an organization, because it can be performed internally within the organization, and can be confined to a small group of evaluators. Yet, this technique is subjective and this is its major drawback. Subjectivity may result from different evaluators producing different evaluation results. A survey technique (for example, using [5]) may reduce subjectivity, however, requires much more resources.

The paper is organized as follows. In Section 2 we introduce and define the evaluation framework. In Section 3 we perform an evaluation over a well-known methodology: GAIA and Section 4 concludes.

2. THE EVALUATION FRAMEWORK

In this paper, we refer to a methodology as the entire set of guidelines and activities: a full lifecycle process; a comprehensive set of concepts and models; a full set of techniques (rules, guidelines, heuristics); a fully delineated set of deliverables; a modeling language; a set of metrics; quality assurance; coding (and other) standards; reuse advice; guidelines for project management. These are each associated with one of four major divisions: concepts and properties, notations and modeling techniques, process, and pragmatics.

2.1 Concepts and Properties

A concept is an abstraction or a notion inferred or derived from specific instances within a problem domain. A property is a special capability or a characteristic. This section deals with the question whether a methodology adheres to the basic notions (concepts and properties) of agents and multi-agent systems. In order to perform such an evaluation we need to define these concepts, since there is no agreement (yet) within the agent community regarding the basic concepts of agent-based systems.

In this paper, we leverage on previous studies (e.g., [5], [11], [12], [14], [16]) and utilize concepts defined there as a basis for our set of concepts. The following are the concepts according to which an agent-oriented methodology should be evaluated:

- **General concepts**

1. **Autonomy:** is the ability of an agent to operate without supervision.
2. **Reactiveness:** is the ability of an agent to respond in a timely manner to changes in the environment.
3. **Proactiveness:** is the ability of an agent to pursue new goals.
4. **Sociality:** is the ability of an agent to interact with other agents by sending and receiving messages, routing these messages, and understanding them.

- **Basic building blocks**

1. **Agent** - A computer program that can accept tasks, can figure out which actions to perform in order to perform these tasks and can actually perform these actions without supervision. It is capable of performing a set of tasks and providing a set of services.
2. **Belief** - A fact that is believed to be true about the world.
3. **Desire** - A fact of which the current value is false and the agent (that owns the desire) would prefer that it be true. Desires within an entity may be contradictory. A widely used specialization of a desire is a goal. The set of goals within an agent should be consistent.
4. **Intention** - A fact that represents the way of realizing a desire. Sometimes referred to as a plan.
5. **Message** - A means of exchanging facts or objects between entities.
6. **Norm** - A guideline that characterizes a society. An agent that wishes to be a member of the society is required to follow all of the norms within. A norm can be referred to as a rule.
7. **Organization** - A group of agents working together to achieve a common purpose. An organization consists of roles that characterize the agents, which are members of the organization.
8. **Protocol** - An ordered set of messages that together define the admissible patterns of a particular type of interaction between entities.
9. **Role** - An abstract representation of an agent's function, service, or identification within a group.
10. **Service:** An interface that is supplied by an agent to the external world. It is a set of tasks that together offer some functional operation. A service may consist of other services.
11. **Society** - A collection of agents and organizations that collaborate to promote their individual goals.
12. **Task** - A piece of work that can be assigned to an agent or performed by it. It may be a function to be performed and may have time constraints. Sometimes referred to as an action.

2.2 Notations and Modeling Techniques

Notations are a technical system of symbols used to represent elements within a system. A modeling technique is a set of models that depict a system at different levels of abstraction and different system's aspects. This section deals with the properties to which methodology's notations and modeling techniques should adhere. The list of these properties is taken from [11].

1. **Accessibility:** is an attribute that refers to the ease, or the simplicity, of understanding and using a method. It enhances both experts and novices capabilities of using a new concept**.**
2. **Analyzability:** is a capability to check the internal consistency or implications of models, or to identify aspects that seem to be unclear, such as the interrelations among seemingly unrelated operations. This capability is usually supported by automatic tools.
3. **Complexity management** (abstraction)**:** is an ability to deal with various levels of abstraction (i.e., various levels of detail). Sometimes, high-level requirements are needed, while in other situations, more detail is required. For example, examining the top level design of a multi-agent system, one would like to understand which agents are within the system, but not necessarily what their attributes and characterizations are. However, when concentrating on a specific task of an agent, the details are much more important than the system architecture.
4. **Executability** (and testability)**:** is a capability of performing a simulation or generating a prototype of at least some aspects of a specification. These would demonstrate possible behaviors of the system being modeled, and help developers determine whether the intended requirements have been expressed.
5. **Expressiveness** (and applicability to multiple domains)**:** is a capability of presenting system concepts that refers to:
   - the structure of the system;
   - the knowledge encapsulated within the system;
   - the system's ontology;
   - the data flow within the system;
   - the control flow within the system;
   - the concurrent activities within the system (and the agents)
   - the resource constraints within the system (e.g., time, CPU and memory);
   - the system's physical architecture;
   - the agents' mobility;
   - the interaction of the system with external systems; and
   - the user interface definitions.
6. **Modularity** (incrementality)**:** is the ability to specify a system in an iterative incremental manner. That is, when new requirements are added it should not affect the existing specifications, but may use them.
7. **Preciseness:** is an attribute of disambiguity. It allows users to avoid misinterpretation of the existing models.

2.3 Process

A development process is a series of actions, changes, and functions that, when performed, result in a working computerized system. This section deals with the process development aspect of a methodology. This aspect is evaluated using the following issues:

1. **Development context:** specifies whether a methodology is useful in creating new software, reengineering or reverse engineering existing software, prototyping, or designing for or with reuse components.
2. **Lifecycle coverage:** Lifecycle coverage of a particular methodology involves ascertaining what elements of software development are dealt with within the methodology. Each methodology may have elements that are useful to several stages of the development life cycle. In this paper, the lifecycle stages are defined as follows:
   - Requirements' gathering is the stage of the lifecycle in which the specification (usually in free text) of the necessities from the system, is done.
   - Analysis is the stage of the lifecycle that describes the outwardly observable characteristics of the system, e.g., functionality, performance, and capacity.
   - Design is the stage of the lifecycle that defines the way in which the system will accomplish its requirements. The models defined in the analysis stage are either refined, or transformed, into design models that depict the logical and the physical nature of the software product.
   - Implementation is the stage of the lifecycle that converts the developed design models into software executable within the system environment. This either involves the hand coding of program units, the automated generation of such code, or the assembly of already built and tested reusable code components from an in-house reusability library.
   - Testing focuses on ensuring that each deliverable from each stage conforms to, and addresses, the stated user requirements.

Having the development stages defined is not sufficient for using a methodology. A methodology should further elaborate the activities within the development lifecycle, in order to provide the user of the methodology with the means of using it properly and efficiently. Providing a detailed description of the various activities during the development lifecycle would enhance the appropriate use a methodology and increase its acceptability as a well-formed engineering approach. Hence, we suggest to examine the process in a more detailed way. These details can be provided by answering the following questions regarding the evaluated methodology:

1. What are the activities within each stage of a methodology? For example, an activity can be the identification of a role, a task, etc. It may consist of heuristics or guidelines helping the developer to achieve his/her goals (in developing the system).
2. What deliverables are generated by the process? This question refers mainly to the documentations. For example, what models are specified and can be delivered to the customer. Another example is weather an acceptance testing plan is required and when it is required.
3. Does the process provide for verification? This question checks whether a methodology has rules for verifying adherence of its deliverables to the requirements.
4. Does the process provide for validation? This question checks whether a methodology has rules for validating that the deliverables of one stage are consistent with its preceding stage.
5. Are quality assurance guidelines supplied?
6. Are there guidelines for project management?

2.4 Pragmatics

Pragmatics refers to dealing with practical aspects of deploying and using a methodology. This section deals with pragmatics of adopting the methodology for a project or within an organization. In particular, the framework suggests examining the following issues:

1. **Resources:** What resources are available in order to support the methodology? Is a textbook available? Are users' groups established? Is training and consulting offered by the vendor and/or third parties? In addition, are automated tools (CASE tools) available in support of the methodology (e.g., graphical editors, code generators, and checkers)? This issue should be examined in order to enable a project/organization aiming at adopting a methodology to check the resources (in terms of training and budget) required and the alternatives for acquiring these.
2. **Required expertise:** What is the required background of those learning the methodology? A distinguishing characteristic of many methodologies is the level of mathematical sophistication required to fully exploit the methodology. Does the methodology assume knowledge in some discipline? This issue should be examined in order to enable a project/organization aiming at adopting a methodology to check whether the qualifications required for using the methodology are met by the candidate users.
3. **Language (paradigm and architecture) suitability:** Is the methodology targeted at a particular implementation language? That is, is the methodology based on concepts from a specific architecture or a programming language? For example, a methodology may be limited to BDI-based applications; it may be oriented towards a specific object-oriented language. This issue should be examined to check whether a methodology adheres to the organization/project infrastructure and knowledge.
4. **Domain applicability:** Is the use of the methodology suitable for a particular application domain (e.g., real-time and information systems)? This issue should be examined to check whether the methodology adheres to the intended problem domain.
5. **Scalability:** Can the methodology, or subsets thereof, be used to handle various application sizes? For example, can it provide a lightweight version for simpler problems? This

issue should be examined to check whether the methodology is appropriate for handling the intended scale of applications within the project/organization.

2.5 Metric

To enable ranking the properties examined in the evaluation process, we propose a scale of 1 to 7 as follows:

1. Indicates that the methodology does not address the property.
2. Indicates that the methodology refers to the property but no details are provided.
3. Indicates that the methodology addresses the property to a limited extent. That is, many issues that are related to the specific property are not addressed.
4. Indicates that the methodology addresses the property, yet some major issues are lacking.
5. Indicates that the methodology addresses the property, however, it lacks one or two major issues related to the specific property.
6. Indicates that the methodology addresses the property with minor deficiencies.
7. Indicates that the methodology fully addresses the property.

In summary, in this section we provided a framework for evaluating agent-oriented methodologies. We divide that framework into four divisions of concepts and properties, notations and modeling techniques, process, and pragmatics. In the proceeding section we demonstrate the use of that framework.

3. EVALUATING GAIA

In this section we evaluate GAIA according to the framework presented in Section 2. We are fully aware of studies that extend GAIA in various aspects such as expressiveness [8] and implementation [10]. However, in the evaluation we performed, we refer only to [17] and [19], as they were written by the methodology designers.

3.1 Concepts and Properties

General concepts

1. **Autonomy:** In GAIA the autonomy is expressed by the fact that the role encapsulates its functionality (i.e., it is responsible for it). This functionality is internal and is not affected by the environment, thus represents the role's autonomy. In addition, in GAIA there is an option to model alternative computational paths, which gives the role (and agents that consist of this role) autonomy in making decisions. The ranking grade is 7.
2. **Reactiveness:** In GAIA the reactiveness is expressed by the liveness properties within the role's responsibilities. However, this does not specify the occurrence of events and the role's reaction to these. The ranking grade is 3.
3. **Proactiveness:** In GAIA the proactiveness is expressed by the liveness properties within the role's responsibilities. The ranking grade is 7.
4. **Sociality:** In GAIA the sociality is expressed within the acquaintance model in which the agent types' interactions are depicted. Further, its sociality is expressed using the organizational structure and rules. The ranking grade is 7.

Basic building blocks

GAIA captures the MAS as a society or an organization. It defines a few abstract concepts as follows: a role, a permission, a responsibility, a protocol and an activity. A role is a definition of functionality within a social group. A permission identifies the resources a role can access. A responsibility is the actual definition of the role functionality. A protocol specifies the way by which a role interacts with another role and an activity is a private action of the role. In addition, GAIA defines a few concrete concepts as follows: an agent type and a service. An agent type is a set of roles and a service is a coherent block of activity, which is defined by pre- and post conditions, inputs, and outputs. Further, GAIA defines a term of an organizational rule to capture the notion of general system constraints.

Examining the coverage of the framework building blocks by GAIA, we found that GAIA addresses most of them, as seen in Table 1. However, the BDI concepts and the knowledge representation as well are not dealt with within GAIA. In addition, GAIA does not support the definition of a service. The ranking grade is 5.

Table 1. The coverage of the framework building blocks within GAIA

Framework building block	GAIA concepts
Agent	Agent type
Belief	
Desire	
Intention	
Message	Protocol
Norm	Organizational rule
Organization	System
Protocol	Protocol
Role	Role
Service	
Society	System
Task	Activity, Responsibility

3.2 Notations and Modeling Techniques

1. **Accessibility:** GAIA models are basically easy to understand and use. Yet, the behavior of the system is introduced via a set of logic expressions which might introduce difficulties to those who would like to understand it. The ranking grade is 5.
2. **Analyzability:** this issue is not dealt with within GAIA. The ranking grade is 1.
3. **Complexity management:** in GAIA, there is no hierarchical presentation or any other mechanism for complexity management. The system description is flat. The ranking grade is 1.
4. **Executability:** this issue is not dealt with within GAIA. The ranking grade is 1.

5. **Expressiveness:** GAIA is expressive and can handle a large variety of systems due to its generic structure. However, GAIA is mostly suitable for small and medium scale systems. This is because of its flatness, which limits the ability to model a large amount of details. In the following we present our analysis regarding the expressiveness of GAIA according to the properties defined in the previous section:
   - the structure of the system is not presented explicitly;
   - the knowledge encapsulated within the system is not presented explicitly;
   - the system's ontology is not dealt with;
   - the data flow within the system is depicted using textual specifications (via the dot and square brackets operators);
   - the control flow within the system is not presented explicitly;
   - the concurrent activities within the system (and the agents) are not presented explicitly;
   - the resource constraints within the system (e.g., time, CPU and memory) are partially specified using the permissions within GAIA;
   - the system's physical architecture is not dealt with;
   - the agents' mobility is not dealt with;
   - the interaction of the system with external systems is not presented explicitly;
   - and the user interface definitions is not dealt with.

   The ranking grade is 4.
6. **Modularity:** GAIA is mostly modular due to its design with some building blocks such as roles, protocols, activities and agent types. In GAIA, one can assign new roles to agents and remove roles with no effect on the internal model of the roles. However, changes within the protocol might cause changes within the internal structure of the role. These result in changes in permissions of the role, hence limits the modularity of GAIA. The ranking grade is 4.
7. **Preciseness:** the liveness and safety properties, which are used for depicting the functionality of a role in a formal way (i.e., for each symbol and notation there is a clear meaning and interpretation), make GAIA accurate and prevent misinterpretation of the modeled functionality. The symbols and notations of each of the other GAIA models have a clear meaning as well. The ranking grade is 5.

3.3 Process

1. **Development context:** GAIA is adequate for the following development contexts: it can be used in creating new software, reengineering and designing systems with reuse components. However, GAIA does not support classical reverse engineering (from code to a model), since it does not address implementation aspects. For the same reason, it cannot be used for prototyping (especially, a rapid one). The ranking grade is 5.
2. **Lifecycle coverage:** lifecycle coverage of GAIA is very limited. It refers only to the analysis and the design stages within the development lifecycle. We found that fact a drawback of GAIA as a methodology, since it would require developers of MAS to adjust the GAIA-based design to the concepts of the target programming language. For example, one may translate the GAIA analysis and design results to UML notations and than use an object-oriented language for implementation. The ranking grade is 3.
3. **Stages' activities within the methodology:** The analysis phase within GAIA includes the identification of the overall goals of the organization and its expected global behavior, the basic skills required by the organization and the basic interactions required for the exploitation of these skills, and the rules that the organization should respect or/and enforce in its global behavior. The design phase within GAIA includes the definition of the organizational structure, the refinement of the roles and the interaction from the analysis phase, the exploitation of well-known organizational patterns, the definition of the agent types that make up the system and assigning roles to them, and the definition of the services that are required for realizing the roles.

   Overall, GAIA provides only a few guidelines for performing the aforementioned activities. The ranking grade is 4.
4. **Methodology deliverables:** each stage supported by GAIA has its own deliverables. The outcomes of the analysis phase are a preliminary role model, a preliminary interaction model, and a set of organizational rules. The outcomes of the design phase are a role model, an interaction model, an organizational structure, a set of organizational rules, a service model, and an agent model. The ranking grade is 7.
5. **Verification and validation:** this issue is not dealt with within GAIA. The ranking grade is 1.
6. **Quality assurance:** this issue is not dealt with within GAIA. The ranking grade is 1.
7. **Project management guidelines:** this issue is not dealt with within GAIA. The ranking grade is 1.

3.4 Pragmatics

1. **Resources:** Although GAIA is well known, there are not much available materials on it (except of the two cited papers). There are no users' groups, nor training or consulting services are offered. Additionally, GAIA does not provide automated tools. The ranking grade is 3.
2. **Required expertise:** GAIA requires a solid background and knowledge in logic and temporal logic. This causes a reduction in its accessibility since many developers do not know or do not want to get familiar with logic (and formal methods). The ranking grade is 2.
3. **Language suitability:** GAIA is not targeted at a specific language. It does not refer to the implementation issues, thus the specification made using GAIA can be implemented in any language. The ranking grade is 7.
4. **Domain applicability:** GAIA, as determined by its developers, is suitable to develop applications with the following characteristics: agents are coarse-grained computational systems, a global goal exists, agents are heterogeneous, the organizational structure is static, the

abilities of the agents are static, and the number of agent types is comparatively small. Yet, GAIA is not suitable for developing applications with dynamic characteristics such as goals generation and changing organizational structure. The ranking grade is 4.

5. **Scalability:** GAIA does not support the use of subsets thereof for system development. Yet, due to its simple structure, it may fit different application sizes. The ranking grade is 4.

3.5 Evaluation Summary

In this section we summarize the evaluation of GAIA. This evaluation demonstrates the use of the proposed framework and the way in which it identifies the strengths and the weaknesses of a methodology. Examining the concepts and properties (as defined by the framework) supported by GAIA, we found that GAIA addresses these to a satisfactory level. Examining the notations and modeling techniques provided by GAIA, we found that it addresses these to a limited extent, mainly due to lack of support in software engineering principles and an insufficient expressiveness of implementation-oriented issues. Examining the process elements provided by GAIA, we found that GAIA provides little details on it, hence further enhancements are required. Finally, examining the pragmatics supported by GAIA, we found that GAIA has a solid theoretical basis but it lacks in providing means to enforce it. Given these results, it seems that there was no intention for GAIA to support stages other than the analysis and design stages, and as a result it lacks in support for other stages.

4. CONCLUSION

In this paper, we propose a framework for evaluating and comparing agent-oriented methodologies. The framework examines the various aspects of a methodology: concepts and properties, notations and modeling techniques, process, and pragmatics.

This framework addresses the problem of evaluating and comparing methodologies. It can be utilized for selecting a methodology for developing an agent-based application. It can additionally be utilized for identifying the advantages and weaknesses of existing agent-oriented methodologies. Identifying these can promote the improvement of such methodologies. Improvement of these may advance the acceptability of agent technology by introducing a mature, well-structured engineering approach.

We demonstrate the use of the framework using a feature analysis technique by performing an evaluation of GAIA – an evolving agent-oriented methodology. GAIA is justifiably considered an advanced agent-oriented methodology, nevertheless, as our study shows, there are several aspects in which the GAIA methodology can be improved, to provide an industry grade methodology. By detecting the shortcomings (and providing the details) of one of the most advanced agent-oriented methodologies, we show that our framework is applicable: it points at the weaknesses of a methodology and thus can promote its improvement.

Although we presented the framework and the feature analysis technique as well-structured and easy to use, we are fully aware of its subjectivity. That is, the ranking grades may vary across evaluators. However, we believe that the overall evaluations resulting from using the proposed framework by several evaluators will be similar due to the well-defined properties and the ranking scale.

Further research is required to evaluate the suggested framework. It may be evaluated with respect to several criteria: accessibility, coverage, adaptability. Accessibility refers to the ease of learning and using the framework; coverage refers to the extent to which the framework addresses the needs of methodologies' evaluation; adaptability refers to the ability to modify and adjust the framework to evaluate domain-specific agent-oriented methodologies. Another research direction could be a comparative evaluation of agent-oriented methodologies, utilizing the proposed framework.

5. REFERENCES

[1] P.Bayer, Marcus Svantesson, "Comparison of Agent-Oriented Methodologies Analysis and Design, MAS-CommonKADS versus Gaia", Blekinge Institute of Technology, Student Workshop on Agent Programming, 2001.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini. “TROPOS: An Agent-Oriented Software Development Methodology”. Accepted to the Journal of Autonomous Agents and Multi-Agent Systems, 2003.

[3] S. Brinkkemper, S. Hong, G. van den Goor, "A formal approach to the comparison of object-oriented analysis and design methodologies", in Proc.of the Twenty-Sixth Hawaii Intl. Conf. on , Vol. 4 , pp. 689-698, Jan 1993.

[4] L. Cernuzzi, G. Rossi, "On the Evaluation of Agent Oriented Methodologies", in Proc. of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, November 2002.

[5] K. H. Dam, M. Winikoff, Survey on Agent-Oriented Methodologies, http://yallara.cs.rmit.edu.au/~kdam/Questionnaire/Questionnaire.html, 2002.

[6] S. A. DeLoach, M. F. Wood, C. H. Sparkman, "Multiagent Systems Engineering", The Intl. Jour. of SE and KE, Vol. 11, No. 3, June 2001.

[7] I. Graham, B. Hederson-Sellers, H. Younessi, “The OPEN Process Specification”, Addison-Wesley, 1997.

[8] T. Juan, A. Pearce, L. Sterling, "ROADMAP: Extending the GIA Methodology for Complex Open Systems", in Proc. of AAMAS'02, pp. 3-10, July 2002.

[9] M. Kumar, “Contrast and comparison of five major Agent Oriented Software Engineering (AOSE) methodologies”, http://students.jmc.ksu.edu/grad/madhukar/www/professional/aosepaper.pdf, 2002.

[10] P. Moraitis, E. Petraki, N. I. Spanoudakis, “Engineering JADE Agents with the Gaia Methodology “, Proc. Int. Workshop on Agents and Software Engineering, 2002.

[11] O. Shehory, A. Sturm, “Evaluation of modeling techniques for agent-based systems”, Agents 2001, pp. 624-631, 2001.

[12] K.Siau and M. Rossi, "Evaluation of Information Modeling Methods – A Review", in Proc. 31 Annual Hawaii International Conference on System Science, pp. 314-322, January 1998.

[13] A. Sturm, D. Dori, O. Shehory, “Single-Model Method for Specifying Multi-Agent Systems”, Proc. Of AAMAS’03, 2003.

[14] Q. N. Tran, G. Low, M. A. Williams, Software Engineering Methodology for Developing Multi-Agent Systems - A Survey, http://129.94.244.146/personal/numi+tran/surveyq.nsf/survey, 2003.

[15] The Object Agency, “A Comparison of Object-Oriented Development Methodologies”, Technical report, http://www.toa.com/smnn?mcr.html, 1995.

[16] M. Winikoff, L. Padgham, and J. Harland, “Simplifying the Development of Intelligent Agents”, Proc of the 14th Australian Joint Conference on Artificial Intelligence (AI'01), pages 557-568, 2001.

[17] M. Wooldridge, N. R. Jennings, D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", Journal of Autonomous Agents and Multi Agent Systems, Vol. 3, No. 3, pp. 285-312, March 2000.

[18] E. Yu, L.M. Cysneiros, "Agent-Oriented Methodologies – Towards A Challenge Exemplar", 4th Intl. Workshop on Agent-Oriented Information Systems (AOIS’02), May 2002.

[19] F. Zambonelli, N. Jennings, M. Wooldridge, “Organizational Rules as an Abstraction for the Analysis and Design of Multiagent Systems”, Intl. Jour. of SE and KE, Vol. 11, No. 4, pp. 303-328, April 2001.