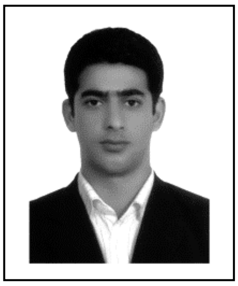
Evaluating GAIAMethodology in Agent-Oriented Software Engineering



Amin Farahbakhsh Tooli
Faculty Of Electrical, Computer and IT Engineering
Qazvin Islamic Azad University
Iran, Qazvin
e-mail: *A.Farahbakhsh@Qiau.ac.ir*

Javad Asadi
Faculty Of Electrical, Computer and IT Engineering
Qazvin Islamic Azad University
Iran, Qazvin
e-mail: *J.Asadi@Qiau.ac.ir*

Paper Reference Number: 1101-777
Name of the Presenter: Amin Farahbakhsh Tooli

Abstract

In this paper, we want to evaluate GAIA methodology by performing a feature analysis which is carried out by evaluating the strengths and weaknesses of it, using an attribute-based evaluation framework. This evaluation framework addresses four major areas of an agent-oriented methodology: concepts, modeling language, process and pragmatics.

Keywords: Agent-Oriented methodology, GAIA, Evaluating Agent-Oriented methodology.

1. Background

The main purpose of this section is to provide some insight into the GAIA software methodology. Hence, the first part of this section provides some background on the agent-oriented approach, including an introduction to agents in Section 1.A.

1.1. Intelligent Agents

"*What's an agent anyway*?" [FON93]. This question was firstly asked in [FON93] and many answers have been proposed to date. According to the Oxford Dictionary, an agent is "one who is authorized to act for or in the place of another". However, among the Artificial Intelligence community the concepts associated with agents, also called software agents or intelligent agents, have been discussed for many years. Even though there are different perspectives on agents [BRA97, FG96], there is an emerging agreement that an agent is "an encapsulated computer system, situated in some environment, and capable of flexible autonomous action in that environment in order to meet its design objectives" [WJ95, page 67]. Examples of software agents can be varied such as robot soccer playing at the RoboCup or intelligent shopping agents helping travelers find airfares or holiday bargains. The applications of agent technologies reside in a wide range of domains, including air traffic control, space exploration, information management, business process management, e-commerce, colonic manufacturing, and defense simulation [LMC03].

Most agent definitions tend to fall in either of the two categories called strong agency and weak agency [WJ95]. Weak agency definitions describe agents with the following characteristics [WJ95]:

- *Situatedness*: Agents are embedded in an environment. They use their sensors to perceive the environment and affect it via their effectors. For example, a robot soccer player is an agent situated in a soccer field. It has different sensors such as cameras for keeping track where the ball and other players are. It also uses several effectors such as its legs or its body for kicking or passing the ball.
- *Autonomy*: Agents can operate and make their own decision on which action they should take, independent of humans or other agents. Hence, agents cannot be directly invoked like objects. Regarding our robot soccer player example, autonomy can be reflected by the fact that when having the possession of the ball, a robot agent is able to decide whether to kick the ball towards the goal or to pass the ball to its teammates. These decisions are made without direct intervention of humans of other robot soccer agents on the field.
- *Reactivity*: Agents can perceive their environment and respond in a timely fashion to changes that occur in it. For instance, when our robot soccer player sees the ball within its control area, it needs to quickly respond to that event by either passing the ball or doing something else with the ball.
- *Pro-activeness*: Agents are pro-active if they have goals that they pursue over time. The ultimate goal of robot soccer players is to win the game by scoring goals and preventing the other team from scoring goals. As a result, most of their actions such as passing the ball to their teammates, kicking for goals, etc. need to contribute toward the achievement of these main purposes.
- *Social ability*: Agents are able to interact with other agents and humans in order to achieve their goals. Relating to our example, social ability requires each robot soccer agent to be able to communicate and coordinate with its teammates.

Strong agents is defined by a similar set of attributes to those of weak agency but the notion is extended, e.g. viewing agents as having mentalistic and intentional notions such as beliefs, desires, goals, intentions and commitments [NWA95]. Additionally, several other features such as mobility (i.e. agents can move around), veracity (i.e. agents are truthful), benevolence (i.e. agents do what they are told to do) are sometimes attributed to strong agency.

The above properties of agents make them flexible and robust entities. Flexibility results from the ability of agents to react to changes and to adapt to the environment by using their social skills (i.e. ability to negotiate with other agents, and ability to take advantage of opportunities). Meanwhile, the capability of exercising choice over their actions and interactions (i.e. autonomy) and pursuing goals (i.e. pro-activeness) improve the agent's robustness. This qualities, flexibility and robustness, are very useful in complex, dynamic, open and failure-prone environments Together with the explosion of distributed information systems including the Internet, these types of environment are becoming more and more popular. As a result, the need to have technologies whose key entities have such intelligent characteristics as agents has been increasing. On the other hand, objects, the fundamental entities in the currently dominant approach, the object-oriented software engineering, cannot offer the same properties as agents do. In fact, the notions of autonomy, flexibility, and pro-activeness can hardly be found in traditional object-oriented approaches [ODE02]. Although agents with these characteristics can be implemented using object-oriented techniques, the standard object-programming model in fact does not specify how to build systems that integrate these types of behavior. Hence, those differences are significant as far as designing and building agent-based systems [WOO02, page 22-23].

2. GAIA methodology

GAIA is one of the first methodologies which is specifically tailored to the analysis and design of agent-based systems [WJ99, WJ00]. Its main purpose is to provide the designers with a modeling framework and several associated techniques to design agent-oriented systems. GAIA separates the process of designing software into two different stages: analysis and design. 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 (Fig 1) depicts the main artifacts of each stage: Role Model and Interaction Model (Analysis), and Agent Model, Services Model, and Acquaintance Model (Design). A detailed description of the process steps which the developers need to follow to build these models is described below.

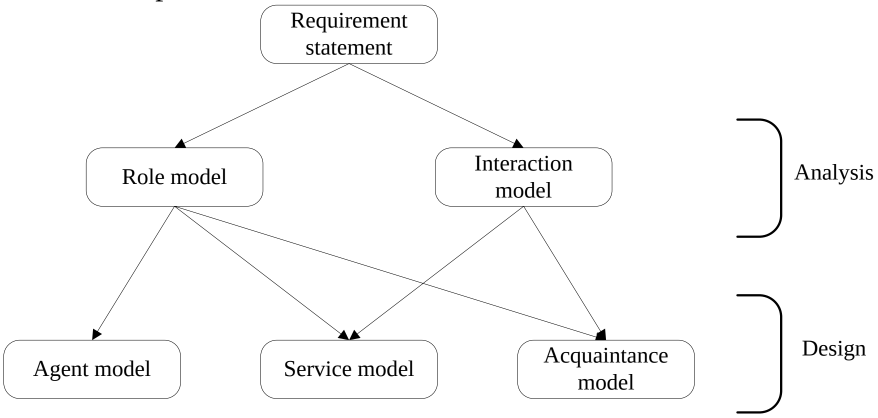


Fig 1: Relationship between GAIA's models (re-draw based on [WJ00])

2.1. Analysis

It is noted that, GAIA assumes the availability of a requirements specification. It means that before beginning the GAIA's development process, the analysts need to have a reasonable understanding of what the system should do. These requirements form the overall objectives of the system which influence the analysis and design phase.

GAIA encourages the developers to view an agent-based system as an organization. The software system organization is similar to a real world organization. It has a certain number of entities playing different roles. For instance, a university organization has several key roles such as administration, teaching, research, students, etc. These roles are played by different people in the university such as managers, lecturers, students, etc. Inspired by that analogy, GAIA guides the designers to the direction of building agent-based system as a process of organizational design.

At the first step of the analysis phase, GAIA requires the analysts to define key roles in the system. At this step, these roles only need to be listed and described in an informal manner. The main purpose of this step understands what roles exist in the system and roughly what they do. GAIA calls the artifact of this step a prototypical roles model.

Different roles in an organization interact with each other to achieve their own goals and also to contribute toward the overall goals of the organization. For example, to fulfill the goal of providing itineraries tailored to specific interest of users, in the context of a Personal Itinerary Planner System, the role that provides the itineraries needs to communicate with the role that keeps track of the interest of users. These interactions need to be defined in the next step of the GAIA analysis phase. The product of this step is the interaction model. This model consists of a set of protocol definitions for each role. Each protocol definition defines the purpose, the initiator, the responder, the inputs, the outputs and the processing during the course of the interaction. The valid sequence of messages involved in a conversation, however, is not required at this stage. Instead, GAIA focuses on the basic nature and purpose of the interaction and abstracts from exact instantiation details.

The final step of GAIA analysis phase involves elaborating the key roles identified in the first step. This process includes the identification of permissions of roles, their responsibilities as well as the protocols and activities in which they participate. This detailed description of a role is depicted by a Role Schemata. A set of Role Schemata forms the Role Model, which is considered as the key artifact of this analysis phase. Responsibility defines the functionality of a role and is divided into two types: liveness properties ("something good happens") and safety properties ("nothing bad happens") [WJ00]. Permissions (i.e. rights) define which resources the agents playing that role can and cannot use when performing a particular action. Activities are "private" actions which do not involve interactions with other roles. Protocols are actions that involve interactions with other roles and are derived from the protocol model built in the previous step.

It is noted that the GAIA analysis process is not purely linear as described. In contrast, the analysts are encouraged to go back to add a new role, new protocols or move forward to add new permissions, activities, etc.

2.2.Design

Having finished the analysis phase, the analysts basically complete building the conceptual model of the system with abstract entities. These entities do not necessarily have any direct realization within the system. They can now move to the second phase (i.e. the Design phase) where those abstract entities are transformed into concrete entities which typically have direct counterparts in the runtime system.

The design stage requires the developers to build three models. First, an agent model which includes various agent types is constructed. Agent types are the counterparts of objects in object-oriented approaches. They are basic design units of an agent-based system and their realization at runtime is agent instances. Agent types in the system under development are defined on the basis of the roles that they play. Therefore, the important feature of this step is to map roles identified in the analysis phase to agent types. A role can be mapped to one or more agent types and vice versa. Some general guidelines are proposed to help this process. For instance, a close relationship between several different roles indicates that they can be grouped together in a single agent type.

The second artifact developed in GAIA's design phase is a service model which depicts the services that each role provides. A service is a "coherent, single block of activity in which an agent will engage" [WJ99]. Each service should be represented by its properties: inputs, outputs, pre-conditions and post-conditions.

Inputs and outputs are derived from the protocol model. Pre-conditions and post-conditions which define the constraints on services are derived from the safety properties of a role. The final model which the designers need to complete is the acquaintance model. It depicts the communication links existing between agent types. It is in fact a directed graph in which nodes represent agent types and arcs show communication pathways.

GAIA does not address implementation and there is no tool support that we, or Michael Wooldridge, one of the authors of GAIA2, are aware of.

3. Evaluating GAIA: Results of the Survey

In this section, we want to evaluate GAIA methodology by performing a feature analysis which is carried out by evaluating the strengths and weaknesses of it, using an attribute-based evaluation framework. This evaluation framework addresses four major areas of an agent-oriented methodology: concepts, modeling language, process and pragmatics.

3.1.Concepts

The results of the evaluation of the GAIA with respect to its concepts are shown in Table 3.1

Internal properties

1. *Autonomy*: As mentioned earlier, autonomy is commonly regarded as one of the key properties of agents. It differentiates agents from other existing entities such as objects. According to the responses from the survey and our assessment, GAIA recognizes that importance. The level of support for autonomy in GAIA is “High”.
2. *Mental attitudes*: GAIA provides weaker support for capturing an agent’s mental attitudes.

Table 3.1: Evaluating GAIA's concepts. L for Low, M for Medium, H for High, N for None.

Concepts & Properties	Result
Autonomy	H
Mental attitudes	L
Proactive	H
Reactive	M
Concurrency	L
Situated	L
Teamwork	N
Protocols	M
Clear concepts	H
Agent-oriented	M

3. *Pro-activeness and reactivity*: Based on the results, it seems that these two attributes are difficult to measure. But they seem to be fairly well supported by GAIA.
4. *Concurrency*: In terms of support for concurrency, GAIA's ratings are “Low”.
5. *Situatedness*: Similar to concurrency, GAIA's rating is “Low” in Situatedness. A model of the environment that is internally used by the agents to represent their environment is not described in the methodology. Although more detailed environment models have been proposed [JPS02], this has not yet been integrated into an industrial-strength methodology.

Social features

1. *Method for cooperation*: There are four cooperation types evaluating in this paper, which are: negotiation, task delegation, multi-agent planning and teamwork. What we are interested in, in this criterion, is the cooperation modes that are clearly supported by the GAIA via provided techniques or models. Only negotiation (i.e. how to manage an acceptable agreement for all agents concerned) and task delegation are directly supported. Multi-agent planning and especially teamwork are not explicitly supported in GAIA methodology.
2. *Teamwork*: Although GAIA support cooperating agents, but it is not support teams of agents in the specific sense of teamwork. As discussed in detail in [CL91], teamwork is the highest level of agent cooperation where all agents in a team work together to achieve a common goal. Members of a team have joint intentions and activities. The behavior of the team looks more like a “single agent with beliefs, goals and intentions of its own, over and above the individual honest” [CL91]. We do not think that GAIA provides support for designing teamwork.
3. *Communication modes*: According to the result of the survey, there was a strong agreement regarding this feature. GAIA provides a wide range of communication modes. More specifically, it supports both direct/indirect and synchronous/asynchronous communication.

4. *Protocols*: Regarding this criterion, based on the respondents including ourselves, GAIA does provide a specific model to represent the protocols but it only captures the interactions between agents at high level. The legitimate sequences of messages exchanged between two agents are not defined in the GAIA protocol model.
5. Communication language: Similar to the Communication modes feature, this one also attracts an agreement among the respondents. Since interactions between agents take place at the knowledge level, GAIA target speech act as a primary communication language.

Overall, according to the results of the survey, the concepts used in the GAIA tend to be clearly explained and understandable.

3.2.Modeling Language

The results of the evaluation of the GAIA with respect to its modeling language is shown in Table 3.2

Table 3.2: Evaluating GAIA's modelling language. A for Agree, N for Neutral, DA for Disagree.

Modeling & Notation	Result
Clear notation	A
Syntax + symbols defined	A
Static + dynamic	A
Adequate & Expressive	N
Different views	DA
Easy to use	A
Easy to learn	A
Semantic defined	A
Consistency checking	N
Traceability	N
Refinement	N
Reusability	DA

Usability criteria

1. *Clarity and understandability*: Two indicators of these two criteria are how clear the notation is and how well the symbols and syntax are defined. As can be seen from Table 3.2, the responders generally agreed that the GAIA's notation were clear and the symbols and syntax are well defined. These indicate the notations provided by GAIA are fairly clear and understandable.
2. *Adequacy and expressiveness*: GAIA's modeling language is not adequate since detailed internal structures of agents are not captured. However, we do not believe that GAIA is a view-oriented methodology. Rather, the methodology represents the target system in terms of different models.
3. *Ease of use*: Overall, most of the respondents agreed that the notations of the GAIA are easy to learn and use. This also relates to the agreement of the understandability and clarity of the notation as discussed above.

Technical criteria

1. *Unambiguity*: Semantics are also well-defined by GAIA. The meaning of symbols and models in GAIA is in fact defined in detail in [WJ00].
2. *Consistency*: GAIA does not appear to support consistency. This response seems to relate the availability of tool support integrated with the methodology. Meanwhile GAIA does not have any tool at all.
3. *Traceability*: Likewise to consistency, GAIA does not support traceability.
4. *Refinement*: Refinement is not supported by GAIA. This reflects the fact that the modeling language of GAIA is integrated into its development process. The process in

fact consists of iterative activities. Developers aren't free to move between phases to add more detail in a constructed model.

5. *Reusability*: According to the result, the responders tended to be disagree in regard of this criterion.

Overall, GAIA has a good modeling language in terms of having clear and understandable notation and being able to express different aspects (e.g. static, dynamic) of the target system. GAIA also has a clear semantic, which reduces the ambiguity of the modeling language. However, several features such as consistency checking and traceability are not supported by GAIA. Meanwhile, GAIA needs some improvement such as integrating these features into the supporting tool.

3.3.Process

The assessment's results of the Process component of GAIA are summarized in Table 3.3 and part of Table 3.4 (for estimating and quality assurance, management decision making guidelines criteria).

The evaluation result of the process steps and life cycle coverage are shown in Table 3.3. The values presented reflect the following:

- 0 indicates no mention is made
- 1 indicates that mention is made but no details are provided.
- 2 indicate that mention is made and at least one example is provided.
- 3 indicate that mention is made, examples are presented, and a process is defined.
- 4 indicate that mention is made, examples are presented, a process is defined, plus heuristics are supplied.

Table 3.3: Evaluating GAIA's process.

Process	Result
Requirements analysis	3
Architectural design	4
Detailed design	1
Implementation	0
Testing & Debugging	0
Deployment	0
Maintenance	0

1. *Development principles*: From the software development life-cycle point of view, GAIA mentions the requirements, architectural design and detailed design to some extent. Implementation, Testing & Debugging, Deployment and Maintenance are not also discussed GAIA. In software engineering generally, a deployment phase usually refer to the activities associated with ensuring that the software product is available for its end users [BRJ98], e.g. packing the software, writing the documentation, etc.

Regarding the software engineering models, we agree with all the responses that GAIA adhere to an iterative development process rather than sequential or waterfall ones. Developers are encouraged to freely move between development phases and steps although there is a tendency in GAIA that specific activity are described in sequence. In terms of the development perspectives, GAIA seems to suit a top-down approach.

2. *Process steps*: The process steps described in the requirements analysis and design phases are also addressed well in GAIA. However, detailed design is not well-documented in GAIA for several reasons such as limited resources and going for generality rather speciality.
3. *Estimating and quality assurance guidelines*: (see Table 3.4) Because of the immaturity of agent-oriented methodologies, issues relating to cost estimating or quality assurance are not addressed in GAIA. It probably relies on the current software engineering practice of these matters.

Overall, GAIA covers requirements analysis, and architectural design. Finally, it needs some improvement in terms of providing estimating and/or software quality assurance guidelines.

3.4.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.

The results of the evaluation of GAIA with respect to its pragmatics are shown in Table 3.4.

Table 3.4: Evaluating GAIA's pragmatics language. A for Agree, DA for Disagree, "-" for no response.

Pragmatics	Result
Quality guidelines	DA
Cost estimation	DA
Management decision	DA
Number apps	0
Real apps	-
Used by non-creators	-
Domain specific	no
Scalable	DA
Distributed	A

Management criteria

1. *Maturity*: Regarding the availability of resources supporting the methodologies, GAIA is in the form of conference papers, journal papers or technical reports and is not published as text books, and GAIA is not supported by any tool also.
2. *Cost*: Regarding the cost of acquiring methodology and tool support, to our knowledge, GAIA is free to access.

Technical criteria

1. *Domain applicability*: The respondents tended to agree that there is no limitation to the application domains in GAIA. These domains are suitable to agent-based system, promising to deliver robust, reliable and autonomous software. Nonetheless, strictly speaking, there are several domain constraints on the basis of models and techniques currently provided by the methodologies. For instance, GAIA not suit systems that allow the possibility of true conflict, e.g. the global system's goal may conflict with those of the system components [JW99]. There are also several assumptions relating to the domain applicability of GAIA such as, the organization structure of the system does not change at run time, or that the ability of agents and the services they provide are unchanged at run-time. Open systems are another area that is not addressed by GAIA.
2. *Dynamic structure and scalability*: Regarding this criterion, most of the respondents stood on a "Disagree" point of view. In our perspective, this issue is explicitly addressed in GAIA. More specifically, it does not tell how to deal with the introduction of new components or modules in an existing system.
3. *Distribution*: Overall, GAIA 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.

4. Conclusion

This article has argued that agent-oriented computing is an appropriate software engineering paradigm for the analysis, design, and development of many contemporary software systems.
while the presented GAIA methodology suggests a rather sequential approach to software development, proceeding linearly from analysis to design and implementation, software development have often to deal with unstable requirements, wrong assumptions, and missing information. This requires the designer to step back from the current activities and, perhaps, to rethink some of the decisions. As it stands, however, there are no explicit structures or processes in GAIA for doing this.
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.

References

BRJ98. G. Booch, J. Rumbaugh, and I. Jacobson. '*The Unified Modelling Language User Guide.*' Addison Wesley, 1998.

BRA97. Jeffrey M. Bradshaw. '*An introduction to software agents.*' In Jeffrey M. Bradshaw, editor, Software Agents, pages 3–46. AAAI Press / The MIT Press, 1997.

CL91. P. R. Cohen and H. J. Levesque. '*Teamwork.*' Nous, 25(4):487–512, 1991.

FER99. Jacques Ferber. '*Multi-agent Systems: An Introduction to Distributed Artificial Intelligence.*' Addison-Wesley, 1999.

FON93. Leonard N. Foner. '*What's an agent anyway? - a sociological case study.*' FTP Report - MIT Media Lab, May 1993.

FG96. S. Franklin and A. Graesser. '*Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents.*' In Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96), volume 1193, Berlin, Germany, 1996. Springer-Verlag.

JW99. N. R. Jennings and Michael Wooldridge. '*Agent-Oriented Software Engineering.*' In Francisco J. Garijo and Magnus Boman, editors, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99), volume 1647, pages 1–7. Springer-Verlag: Heidelberg, Germany, 30– 2 1999.

JPS02. T. Juan, A. Pearce, and L. Sterling. '*Roadmap: Extending the GAIA methodology for complex open systems.*' In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), Bologna, Italy, July 2002.

LMC03. 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.

NWA95. H. S. Nwana. '*Software agents: An overview.*' Knowledge Engineering Review, 11(2):205– 244, 1995.

ODE02. James Odell. '*Objects and agents compared.*' Journal of Object Technology, 1(1):41–53, 2002.

WJ95. M. Wooldridge and N. R. Jennings. '*Intelligent agents: Theory and practice.*' The Knowledge Engineering Review, 10(2):115–152, 1995.

WJ99. 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), 1999.

WJ00. 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.

WOO02. M. J. Wooldridge. '*An Introduction to Multi-Agent Systems.*' John Wiley & Sons, 2002.