Requirements for Multi-Agent Systems

Carla Silva, Rosa Pinto, Jaelson Castro, Patrícia Tedesco

Universidade Federal de Pernambuco - CIn - Recife (PE) – Brasil
Caixa Postal 7851
50732-970 - Recife - PE – Brasil
{ctlls, rccp, jbc, pcart}@cin.ufpe.br

Abstract. Autonomous agents are beginning to be used as a software paradigm, because of their potential to build more powerful and flexible complex systems. To achieve such benefits a standard definition of is agenthood is necessary. In doing so, agent-oriented software engineering will not only be able to develop standardised processes for building agent-based systems but also be able to better as evaluate existing agent-oriented methodologies. This paper presents a set of requirements for agent-oriented systems and the relationships between them using the NFR framework. In order to exemplify the use of the defined criteria, we present a comparative study of two important agent-oriented methodologies.

1 Introduction

Agent orientation has the potential to become a mainstream software engineering paradigm in response to the increasing complexity of today's software systems. In particular, these systems are required to operate in complex – distributed, large, open, dynamic, unpredictable, heterogeneous, and highly interactive – application environments [1].

Agenthood offers a higher level of abstraction in thinking about software systems features and behaviour. Hence, it seems natural to build complex software systems in terms of agents and multi-agent technology. Agent orientation is beginning to be used in industrial and commercial applications, ceasing to exist only in the academic environment. The usage of this technology in industry has demonstrated that agent oriented techniques lead to improvement of distributed complex system development. However, the benefits promised by the agents paradigm cannot be fully achieved yet because although many different perspectives of agency have been described and discussed, there is no universally accepted definition of what exactly determines agenthood. Therefore, there is no standardised development process to build agent-oriented applications.

This paper is an attempt to establish the requirements for multi-agent systems, i.e., to determine which concepts and features really need to be defined in order to build an agent-oriented software system. After that, we analyse the most well known agent-oriented methodologies based on the requirements we have established. Such analysis enables us to point out the most suitable one for a specific agent-based application. It

is worth remarking that we use the NFR framework [2] to express the relationships among the requirements. The NFR is adequate to this situation because it makes explicit the relationships between non-functional requirements and the design decisions intended to implement them.

The paper is organised as follows: section 2 reviews the some current evolving agent-oriented methodologies. Section 3 presents our contribution by establishing the requirements for multi-agent systems. Section 4 exposes a comparison between two agent-oriented methodologies based on our proposal. Section 5 discusses related work. Finally, section 6 summarizes our work and points out urgent and still open issues in agent-oriented software engineering.

2 Agent-Oriented Software Engineering and Methodologies

Agent-oriented software engineering is concerned with the use of agents in the development of complex distributed systems especially in open and dynamic environments. Agents provide a natural and elegant means to manage complexity and interactions. The agent abstraction may be applied not just to represent technological components of implemented systems, but also to the modelling and design of complex systems that may be implemented in the most appropriate fashion.

Agents provide designers and developers with a way of structuring an application around autonomous, communicative elements. In order to support this view of systems development, certain tools and techniques need to be introduced. For example, methodologies to guide analysis and design are required; agent architectures are needed for the design of individual components, and supporting infrastructure (including more general, current technologies, such as web services) must be integrated.

The increasing interest in software agents and multi-agent systems has recently led to the development of new methodologies based on agent concepts. Modelling languages and methodologies such as GAIA [3], KGR [4], AUML [5], MaSE [6] and Tropos [7] have become the focus of the emerging area of agent-oriented software engineering. These methodologies propose different approaches in using agent concepts and techniques at various stages during the software development lifecycle.

Two main trends are considered in multi-agent system design: extending software engineering (or knowledge engineering) methodologies, or extending specific agent methodologies. KGR [4], and Tropos [7] are examples of the former case, whereas GAIA [3] and MaSE [6] exemplify the latter.

KGR [4] focus on two viewpoints. The external viewpoint describes the social system structure and dynamics. It includes an Agent Model and an Interaction Model. The internal viewpoint is composed of three models: the Belief Model, the Goal Model, and the Plan Model. These models specify how an agent perceives the environment and how it chooses its actions based on this perception.

AUML [5] is an analysis and design methodology, which extends UML to represent agents. It provides extensions to UML by adding a three-layer representation for agent interactions protocols (AIP), which define communication protocol as ‘an al-

lowed sequence of messages between agents, and the constraints on the contents of these messages'.

Tropos [7] is a development framework founded on concepts used to model early requirements. The proposal adopts Eric Yu's *i* modelling framework* [14], which offers the notions of actor, goal and (actor) dependency, and uses these as a foundation to model early and late requirements, architectural and detailed design.

GAIA [3] makes an important distinction between the analysis (dealing with *abstract* concepts) and the design (dealing with *concrete* concepts) processes, and provides several models to be used at each phase. In essence it constructs a society of agents, defining the role and capabilities of each individual agent, and the way the society of agents is structured.

MaSE [6] takes an initial system specification, and produces a set of formal design documents in a graphically based style. The primary focus is to guide a designer through the software lifecycle from a prose specification to an implemented agent system.

Although many methodologies for developing agent-based systems have been proposed and developed, it is difficult to select a specific methodology or to determine which are the advantages of each one. Comparing methodologies is often difficult, since they usually address different properties of a software agent. In an attempt to solve this problem, we propose a set of requirements for developing agent-oriented systems, discussed next section. We also describe the relationships between these requirements.

3 Requirements for Designing Multi-Agent Systems

Some of the properties of software systems emerge from the combination of its parts. These emergent properties will surely be a matter of accident, not design, if the non-functional requirements (system qualities) are not specified in advance. This happens because non-functional requirements (NFRs) impact on the rest of software development, especially during the design phase. Yet they are hard to deal with since they are hard to quantify, and often can conflict each other. During system's design, such non-functional requirements appear in design tradeoffs when designers need to decide upon particular structural or behavioural aspects of the system [9]. Specifying NFRs for agent based systems is yet more critical, because such systems present, in addition to the problems of traditionally distributed and concurrent systems, the difficulties that arise on from enabling flexibility and sophisticated interactions [10].

Based on our bibliographical research [1][11][12][13], together with our Software Engineering experience, we establish the following agent properties as non-functional requirements for multi-agent systems:

– *Autonomy*: ability of the software to act independently without direct intervention from humans or other agents. Active autonomous entities are not necessarily compliant with external demands or desires [14].

- *Deliberativity*: A deliberative agent is one whose actions are not driven solely by events changes in its environment. Its actions are decided by considering both information from environment and information about previous experiences, generating goals and acting rationally to achieve them.
- *Reactivity*: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET), and respond in a timely fashion to changes that may occur.
- *Organisation*: the arrangement of relationships between agents that produces a unit endowed with qualities not apprehended at the level of the individual [1].
- *Sociality*: ability to participate in multiple relationships, interacting with a number of other agents, at the same time or at different times [14].
- *Interaction*: ability to communicate with the environment and other agents.
- *Coordinatition*: ability to perform some activity in a shared environment with other agents, determining goals they share and common tasks, avoiding unnecessary conflicts and pooling knowledge and evidence [1].
- *Cooperation*: ability to interact with other agents to achieve a common purpose; nonantagonistic agents that succeed or fail together.
- *Competition*: ability to interact with other agents where the success of one agent implies the failure of others (the opposite of cooperation).
- *Negotiation*: ability to interact with other agents in order to reach an agreement about some matter. It involves the exchange of information, the relaxation of initial goals, mutual concessions, lies or threats [15].

In this paper we show the NFR framework [2] can be useful to describe non-functional requirements for agent-oriented systems as well as the design decisions to implement these properties,. The analysis involves refining these properties, represented as softgoals, to sub-goals that are more specific and more precise and then evaluating design alternatives. This framework was one significant step in making the relationships between non-functional requirements and intended decisions explicit. The design decisions are represented as operationalised softgoals and are linked to NFRs though contribution links.

In Figure 1, we depict a SIG (Softgoal Interchange Graph) for agent NFRs. Throughout the paper we use the NFR framework notations to indicate how the operationalised softgoals satisfy a given NFR. Such notation includes terms like: some +, help, make. These model some/positive, partial/positive, sufficient/positive, contributions, respectively.

From our point of view software agents have autonomy and are social. They interact to maintain a relationship with each other in order to ensure the achievement of their goals. In particular, the arrangement of relationships between agents produces a unit endowed with qualities not apprehended at the level of the individual – an agent organisation. In each organisation, an agent can play one or more roles. A role is what the agent is expected to do in the organisation: both in cooperation with the other agents and in respect of the organisation itself [17]. A role also entails duties and privileges to the agent. However, organisation requires a certain amount of order among entities which may be heterogeneous and this order contributes to the coher-

ence of the whole, i.e., agents need to coordinate their action [1]. Without coordination, any benefits of interaction vanish and the group of agents quickly degenerates into a collection of individuals with a chaotic behaviour [15]. In this context, Sociability is AND-decomposed into Interaction and Organisation.

There may be conflicts or potential conflicts arising from multiple relationships that an agent engages in [14]. Therefore, agents need to interact with each other in order to both reach an agreement about some matter and cooperate in activities present in organisation. Also, another kind of interaction happens between antagonist agents, named competition. From this perspective, interaction can be a simple communication between agents, a competition or a cooperation involving coordination of activities and a complex negotiation to solve conflicts and obtain consensus about something.

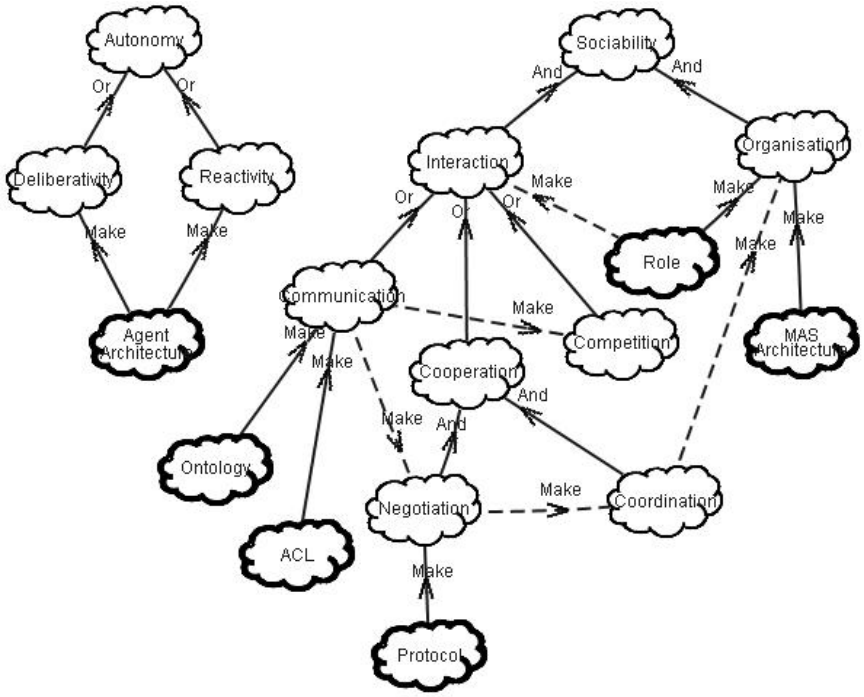

Figure 1. SIG for multi-agent systems

In Figure 1, Interaction is OR-decomposed into Communication, Cooperation and Competition. Moreover, to achieve cooperation agents need to coordinate and negotiate with each other. Hence, Cooperation is AND-decomposed into Negotiation and Coordination.

Autonomy is OR-decomposed into Deliberativity and Reactivity. Autonomy means the agent can initiate an action instead of passively waiting to be manipulated by an outside executor. Deliberativity states that performance of an agent is in a goal-driven

manner. Reactivity emphasises that as the environment changes this also affects the agent's behaviour as well as its internal goal. So an agent can present deliberative or reactive autonomy.

Greater autonomy implies more powerful software, which is likely to be more challenging to design and implement [14]. In fact, there is both reactive and deliberative agent architecture available in literature that can address the design of these kinds of autonomy in a multi-agent system [18][19]. These architectures are represents in the SIG as operationalised softgoals (bold clouds).

To achieve negotiation, agents need to use a protocol, which defines the legal proposals that agents can make. To enable communication agents is necessary both an ontology (i.e., an explicit and precise description of domain concepts and relationships among them) and an agent communication language (ACL), such as KQML [19] or FIPA-ACL [21]. The protocol and ontology together with ACL are design decisions (operationalised softgoals in SIG) necessary to implement negotiation and communication, respectively, in multi-agent systems.

To model an organisation, we need to define a structured description of elements from which multi-agent software systems are built, i.e., a software architecture. Some methodologies have already defined several high-level organisational patterns for multi-agents architecture [21] (operationalised softgoal in SIG). An agent plays a given role (operationalised softgoals in SIG) and has a well-defined position in the organisation. Hence, it is committed to certain interaction protocols with the other agents in the organisation. Therefore, the concept of inter-agent interaction is strictly related to the agent's role (correlation links).

Some NFRs can contribute or conflict with each other (correlation links). Communication among agents, for example, enables competition and negotiation. Negotiation is needed into interaction in a multi-agent system to enable Coordination. Coordination, in turn, contributes to the coherence of the Organisation.

4 Comparing Agent-oriented Methodologies

Methodologies may differ in their objectives and underlying premises as well as the way they deal with the non-functional requirements discussed in the previous session. In the sequel we first describe two well-known methodologies, namely GAIA and Tropos. Then we compare them according to the requirements for multi-agent systems described in this work.

4.1 Tropos

Tropos proposes a software development methodology and a development framework which are founded on concepts used to model early requirements and complements proposals for agent-oriented programming platforms [7]. Tropos supports four phases of software development:

- Early requirements, concerned with the understanding of a problem by studying an organisational setting; the output is an organisational model which includes relevant actors, their goals and dependencies.
- Late requirements, in which the system-to-be is described within its operational environment, along with relevant functions and qualities.
- Architectural design, in which the system's global architecture is defined in terms of subsystems, interconnected through data, control and dependencies.
- Detailed design, in which behaviour of each architectural component is defined in further detail.

To support modelling and analysis during the initial phases, Tropos adopts the concepts offered by *i** [8], a modelling framework offering concepts such as actor (actors can be agents, positions or roles), as well as social dependencies among actors, including goal, softgoal, task and resource dependencies. This means that both the system's environment and the system itself are seen as organizations of actors, each having goals to be fulfilled and each relying on other actors to help them with goal fulfillment.

As shown in Figure 2, actors are represented as circles; dependums -- goals, softgoals, tasks and resources -- are respectively represented as ovals, clouds, hexagons and rectangles; and dependencies have the form depender⇒dependum⇒dependee. Hence, in Tropos we have the following concepts:

- Actor: An actor is an active entity that carries out actions to achieve goals by exercising its know-how.
- Dependency: A dependency describes an intentional relationship between two actors, i.e., an "agreement" (called dependum) between two actors: the depender and the dependee, where one actor (depender) depends on another actor (dependee) on something (dependum).
  - Depender: The depender is the depending actor.
  - Dependee: The dependee is the actor who is depended upon.
  - Dependum: The dependum is the type of the dependency and describes the nature of the agreement.
- Goal: A goal is a condition or state of affairs in the world that the stakeholders would like to achieve. How the goal is to be achieved is not specified, allowing alternatives to be considered.
- Softgoal: A softgoal is a condition or state of affairs in the world that the actor would like to achieve, but unlike in the concept of (hard) goal, there are no clear-cut criteria for whether the condition is achieved, and it is up to subjective judgment and interpretation of the developer to judge whether a particular state of affairs in fact achieves sufficiently the stated softgoal.
- Resource: A resource is an (physical or informational) entity, with which the main concern is whether it is available.
- Task: A task specifies a particular way of doing something. Tasks can also be seen as the solutions in the target system, which will satisfy the softgoals (operationalisations). These solutions provide operations, processes, data representations, structur-

ing, constraints and agents in the target system to meet the needs stated in the goals and softgoals.

To support modelling and design during the later phases, Tropos proposes to adopt existing agent communication languages like FIPA-ACL [20] or KQML [20], message transportation mechanisms and other concepts and tools. One possibility is to adopt extensions to UML [23], like AUML, the Agent Unified Modelling Language [5] proposed by the Foundation for Physical Intelligent Agents (FIPA) [23] and the OMG Agent Work group.

This methodology is based on the premise that in order to build software that operates within a dynamic environment, one needs to analyse and model explicitly that environment in terms of "actors", their goals and dependencies on other actors.

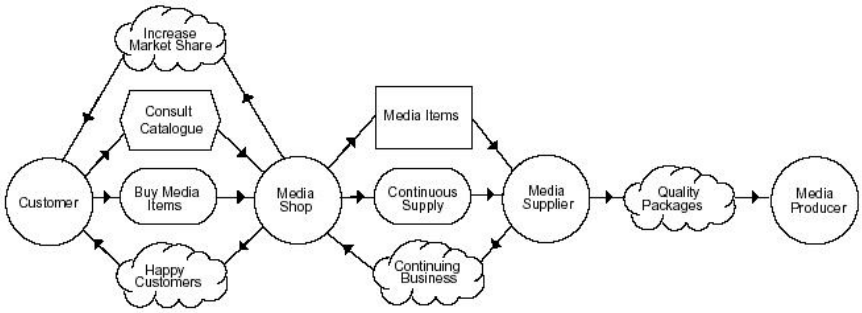

Figure 2. i* model for a media shop

4.2 GAIA

GAIA [3] is intended to allow an analyst to go systematically from a statement of requirements to a design that is so detailed so that it can be implemented directly. Analysis and design can be thought of as a process of developing increasingly detailed models of the system to be constructed (Figure 3).

GAIA provides an agent-specific set of concepts through which a software engineer can understand and model complex systems. In particular, GAIA encourages a developer to think of building agent-based systems as a process of organisational design. The main GAIA concepts can be divided into two categories: abstract and concrete. Abstract entities are those used during Analysis to conceptualise the system, but which do not necessarily have any direct realisation within the system. Concrete entities, in contrast, are used within the Design process, and will typically have direct counterparts in the run-time system.

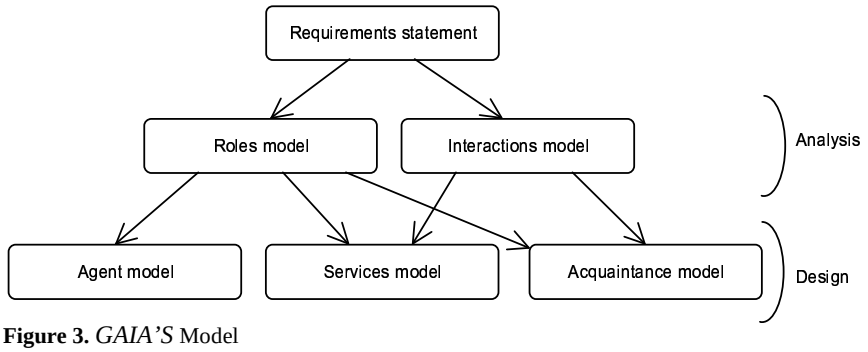

Figure 3. *GAIA'S* Model

4.2.1 Analysis

The objective of the analysis stage is to develop an understanding of the system and its structure (without referring to any implementation detail). This understanding is captured in the system's organisation. An organisation can be seen as a collection of roles, and that take part in systematic, institutionalised patterns of interactions with other roles (as depicted in Figure 4).

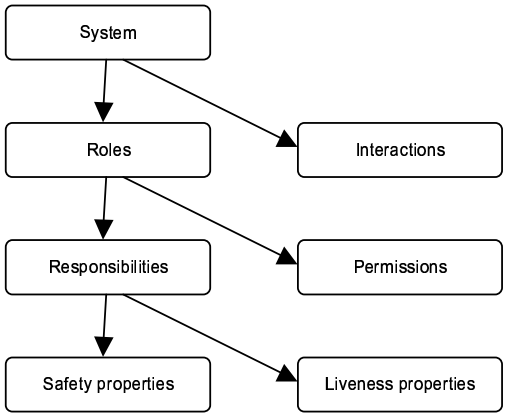

Figure 4. Analysis Concepts

The most abstract entity is the system with the meaning of society or organisation. The idea of a system as a society is useful when thinking about the next level in the concept hierarchy: roles.

A role is defined by four attributes: responsibilities, permissions, activities, and protocols. Responsibilities determine functionality and, as such, are perhaps the key attribute associated with a role. Responsibilities are divided into two types: liveness properties and safety properties. The former intuitively state that something good happens. They describe those states of affairs that an agent must bring about, given

certain environmental conditions. In contrast, safety properties are invariants. In other words, a safety property states that nothing bad happens (i.e., that an acceptable state of affairs is maintained across all states of execution). In order to realise responsibilities, a role has a set of permissions, which are the rights associated with a role. They identify the resources that are available to that role in order to realise its responsibilities. Permissions tend to be information resources.

The activities of a role are computations associated with it that may be carried out by the agent itself. Finally, a role is also identified with a number of protocols, which define the way that it can interact with other roles. Thus, the organisation model in two further models: the roles model and the interaction model.

- The roles model identifies the key roles in the system, and has the characteristics described above.
- The interaction model nominal links between roles. This model consists of a set of protocol definitions, one for each type of inter-role interaction. Here a protocol can be viewed as an institutionalised pattern of interaction. That is, a pattern of interaction that has been formally defined and abstracted away from any particular sequence of execution steps.

4.2.2 Design

The aim in GAIA is to transform the analysis models into a sufficiently low level of abstraction that traditional design techniques (including object-oriented techniques) may be applied in order to implement agents. In other words, GAIA is concerned with how a society of agents cooperate to realise the system-level goals, and what is required of each individual agent in order to do this. Actually how an agent performs its services is beyond the scope of GAIA, and will depend on the particular application domain.

The GAIA design process involves generating three models (see Figure 3). The agent model identifies the agent types that will make up the system, and the agent instances that will be instantiated from these types. The services model identifies the main services that are required to realise the agent's role. Finally, the acquaintance model documents the lines of communication between the different agents.

GAIA is founded on the view of a multi-agent system as a computational organisation consisting of various interacting roles. GAIA deals with both the macro (societal) level and the micro (agent) level aspects of design. It represents an advance over previous agent-oriented methodologies in that it is neutral with respect to both the target domain and the agent architecture.

4.3 Comparing GAIA and Tropos

We briefly compare GAIA and Tropos according to the requirements for multi-agent systems described in this work. We have only considered a subset of the requirements depicted in figure 1 because they are the ones found in the case studies existing on literature of both methodologies. As expected each methodology addresses these properties differently:

- *Organisation*: In GAIA, organisational rules, organisational structures and organisational patterns play a primary role in the analysis and design of such Multi-Agent Software (MAS). Organisational rules express relationships and constraints between roles, between protocols, and between roles and protocols, that can drive the identification of the organisational structure. Organisational rules express general, global requirements for the proper instantiation and execution of a MAS. An organisational structure defines the specific class (among the many possibilities) of organisation and control regime to which the agents and roles have to conform in order for the whole MAS to work efficiently and according to its specified requirements. Organisational patterns express pre-defined and ubiquitous organisational structures that can be re-used from system to system [17]. In Tropos, a software system is structured in terms of a social organization of coordinated autonomous components that interact in order to achieve specific and possibly common goals. Hence, organizational architectural styles [21] for agent, cooperative, dynamic and distributed applications have been defined to guide the design of the system architecture. The purpose is to reduce as much as possible the impedance mismatch between the system and its environment.
- *Interaction*: In GAIA, organisational role models precisely describe all the roles that constitute the computational organisation; in terms of their functionalities, activities and responsibilities, as well as in terms of their interaction protocols and patterns. The interactions model captures the protocols that are associated with the roles. GAIA also defines global rules ("coordination laws") to specify the behaviour and the interaction of agent ensembles. Thus, all interactions have to occur via specific "coordination media", whose internal behaviour can be programmed so as to implement specific policies for governing agent interactions. However, only recently coordination models have been recognized as useful abstractions upon which to define methodologies for the analysis and design of open agent systems. In Tropos, interaction is represented though AUML's sequence diagrams and FIPA-ACL. In particular, a customisation of the FIFA Contract Net Protocol [5] is used to describe a communication pattern among agents, as well as constraints on the contents of the messages they exchange.
- *Autonomy*: In both GAIA and Tropos autonomy is expressed by the fact that a role encapsulates its functionality (i.e, it is responsible for it execution). This functionality is internal and is not affected by the environment, thus represents the role's autonomy (and agents that consist of this role).
- *Sociability*: In both GAIA and Tropos the sociability is expressed using the organisational structures and organisational styles respectively.
- *Reactivity and Deliberativity*: In GAIA deliberativity and reactivity are expressed by the liveness properties (i.e., properties that the system must guarantee to enable "something good" happens in the organisation) within the role's responsibilities. However, this does not specify the occurrence of events and the role's reaction to these [11]. In Tropos, reactivity and deliberativity are expressed though AUML's plan diagram. This diagram depicts the internal behaviour of an agent participating of a specific interaction protocol.

In order to summarise our findings from this comparison, table 1 details the different degrees of satisfaction of the NFRs for multi-agent systems by both GAIA and TROPOS.

Table 1. Agent-based methodologies versus NFRs for multi-agent systems

Requirements	GAIA	Tropos
Organisation	make	make
Interaction	make	help
Autonomy	make	make
Sociability	make	make
Reactivity	some +	help
Deliberativity	some +	help

According to Table 1, both GAIA and Tropos fully satisfy the Organisation, Autonomy and Sociability aspects of a multi-agent system, as we have argued above. GAIA fully satisfies the Interaction property of a multi-agent system, while Tropos just satisfies it partially since Gaia interaction models denote the process in more details than the ones from Tropos. Reactivity and Deliberativity features are partially satisfied by Tropos, where we have state diagrams to model agents´ behaviour, but weakly satisfied by GAIA since it doesn't support diagrams to model events like Tropos does.

6 Related Work

Many different perspectives of agency have been described and discussed, and there is no consensus about the proper definition of what exactly determines agenthood. Even so, we can find some explanation that seeks to contribute to a clarification of the concept of agent in [25,24]. Introductory texts about software agents are [26,26]. Well written course-level texts on computational agency are [27, Chapter 2] and [28]. Finally, books that broadly cover agent and multi-agent technology are [29,14].

Although there has not been much work in comparing agent-oriented methodologies, a framework to carry out an evaluation of agent-oriented analysis and design modelling methods has been proposed by [30]. The significance of the framework is the construction of an attribute tree, where each node of the tree represents a software engineering criterion or characteristic of agent-based system. In [30] a comparison of agent-oriented methodologies is performed based upon an attribute-based framework which addresses four major areas: concepts, modelling language, process and pragmatics. In [11] a framework for evaluating and comparing agent-oriented methodologies is proposed, focusing on four major aspects of a methodology: concepts and properties, notations and modelling techniques, process and pragmatics.

7 Conclusions and Further Work

In order to establish the key properties a Multi-Agent System has to present, we have attempted to define a set of requirements for agent-oriented software and the relationships between them by using the NFR framework. Properties like autonomy, deliberativity, reactivity, sociability, coordination, organisation and interaction have been described, analysed and related to each other.

Several agent-oriented methodologies have been proposed in the literature. However, a crucial step is to understand the relationships between these various methodologies and particularly to understand the main properties addresses by them. Comparing methodologies identifies the strengths and the weaknesses of existing agent-oriented methodologies, leading to their improvement.

One of the contributions of our work is to provide a comprehensive catalogue of interrelated agent properties (described in terms of NFR diagrams) which can be used to help organisations to select a methodology suitable for the development of agent-based applications. It can also help researchers to examine the similarity and the differences among existing agent-oriented methodologies. Thus, evaluating methodologies plays an important role in improving them and in developing the next-generation of agent-oriented methodologies.

Deciding what methodology is the best one will depend on which requirements are considered more important for a specific agent-based application. A correlation catalogue evaluating several agent-oriented methodologies can certainly help the designers decision in this aspect.

A next step attempting to standardize agent-oriented software engineering is comparing other existing methodologies with respect to the support of non-functional requirements described in this work. Moreover, we intend to exemplify the presence of these requirements into a case study of a multi-agent system for traffic simulation we are currently working on.

References

[1]Weiss, G. Multiagent systems: a modern approach to distributed artificial intelligence. Second Printing. Massachusetts Institute of Technology, 2300.

[2]Chung, L., Nixon, B. A., Yu, E., Mylopoulos, J. Non-functional requirements in software engineering. Kluwer Academic Press, Boston et al., 2300.

[3]Wooldridge, M., Jennings, N. R.., Kinny, D. The GAIA Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agents System, 3(3): 285-315, 2300.

[4]Kinny, D., Georgeff, M., Rao, A. A Methodology and Modelling Technique for Systems of BDI Agents, in W. Van Der Velde and J. Perram, editors., Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World MAAMAW'96, (LNAI Volume 1038). Springer-Verlag: Heidelberg, Germany, 2296.

[5]Odell, J., Parunak, H. V. D., Bauer, B. Extending UML for agents. In Proc. of the 2nd Int. Bi-Conference Workshop on Agent-Oriented Information Systems, AOIS'00, pages 3–20, Austin, USA, July 2300.

[6]Wood, M.F., DeLoach, S. A. An Overview Multiagent System Engineering Methodology. In Agent-Oriented Software Engineering – Proceedings of the First International Workshop on Agent-Oriented Software Engineering, 10 th June 2300, Limerick, Ireland. P. Ciancarini, M. Wooldridge, (Eds.) Lecture Notes in Computer Science. Vol. 2257, Springer Verlag, Berlin, January 2301.

[7]Castro, J. Kolp, M., Mylopoulos, J. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems Journal, Elsevier, 2302. Vol 27, pp. 365-89.

[8]Yu., E. Modelling Strategic Relationships for Process Reengineering. Ph.D. thesis, Department of Computer Science, University of Toronto, Canada (2295).

[9]Gross, D., Yu, E. Evolving System Architecture to Meet Changing Business Goals: an Agent and Goal-Oriented Approach ICSE-2301 Workshop: From Software Requirements to Architectures (STRAW 2301) May 2301, Toronto, Canada. pp. 16-21.

[10]Zambonelli, F., Jennings, N., Ornicini, A., Wooldridge, M. Agent Oriented Software Engineering for Internet Applications, Published as Chapter 16 in the book: Coordination of Internet Agents: Models, Technologies and Applications, F. Zambonelli, M. Klusch, R. Tolksdorf (Eds.), Springer (2300).

[11]Sturm, A., Shehory, O. A Framework for Evaluating Agent-Oriented Methodologies. Fifth International Bi-Conference Workshop on AGENT-ORIENTED INFORMATION SYSTEMS (AOIS-2303) 17 July 2303, Melbourne, Australia, at AAMAS'03.

[12]Bradshaw, J. M. An introduction to software agents. In J. M. Bradshaw, editor, Software Agents, pages 3-46. AAAI Pres/The MIT Press, 2297.

[13]Russel, S. J., Norvig, P. Artificial Intelligence. A modern Approach. Prentice Hall, Engle

[14]Yu, E. Agent-oriented modelling: software versus world. In M. J. Wooldridge, G. Weiss, and P. Ciancarini, editors, Agent-roeinted software engineering. Proceedings of the Second International Workshop (AOSE-2301), Lecture Notes in Artificial Intelligence, Vol. 2424. Springer-Verlag, 2302.

[15]Green, S., Hurst, L., Nangle, B., Cunningham, P., Somers, F., Evans, R. Software Agents: A Review. Technical report. Trinity Collega, Dublin, Ireland, May 2297.

[16]Zambonelli, F., Jennings, N.R., Wooldridge, M. Organisational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini and M.J. Wooldridge, editors, Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2300), Lecture Notes in Articial Intelligence, Vol. 2257, pages 255{262. Springer-Verlag, 2301.

[17]Muller, J. P. The right agent (architecture) to do the right thing. I J.P. Muller, M.P Singh, and A. S. Rao, editors, Intelligent Agents V, Lecture Notes in Artificial Inteligence, Vol. 1855, pages 214-246. Springer-Verlag. Berlin et al., 2299.

[18]Muller, J. P. Control architectures for autonomous and interacting agents: A survey. In L. Cavedon, L. Rao, and W. Wobcke, editors, Intelligent Agent Systems: Theoretical and Practical Issues, Lecture Notes in Artificial Intelligence, Vol. 1539. Springer-Verlag et al., 2296.

[19]Finin, T., Labrou, Y., Mayfiled, J. KQML as an agent communication language. In J. M. Bradshaw, editor, Software Agents, pages 291-319. AAAI Press/The MIT Press, 2297.

[20]Labrou, Y., Finin, T., Peng, Y.: The current landscape of agent communication languages, Intell. Systems 17 (2) (2299) 45-52.

[21]Kolp, M., Castro, J., Mylopoulos, J. A social organization perspective on software architectures. In Proc. of the 1st Int. Workshop From Software Requirements to Architectures. STRAW'01, Toronto, Canada (2301) 5–15.

[22]Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language – Reference Manual. Addison Wesley (2299).

[23]FIPA. FIPA (Foundation for Intelligent Agents), http://www.fipa.org, 2299.

wood Cliffs, New Jersey, 2295.

[24]Franklin, S., Graesser, A. Is it an agent, or just a program?: A taxomony for autonomous agents. In J.P. Muller, M.J. Wooldridge, and N.R. Jennings, editors, Intelligent Agents III, Lecture Notes in Artificial in Artificial Intelligence. Vol. 1523, pages 21-36. Springer-Verlag, Berlin et al., 2297.

[25]Wooldridge, M. J., Jennigs, N.R. Agent theories, architectures, and languages: A survey. In M.J. Wooldridge and N.R. Jennings, editors, Intelligent Agents, Lecture Notes in Artificial in Artificial Intelligence, Vol. 890, pages 1-39. Springer-Verlag, Berlin et al.,2295.

[26]Nwana, H. S. Software Agents: An overview. The knowledge Engineering Review, 14(3): 235-244, 2296.

[27]Wooldridge, M. J. Intelligent agents. In G. Weiss, editor, Multiagents Systems, pages 27-77. The MIT Press, Cambridge et al, 2299.

[28]Bradshaw, J. M. Handbook of agent technology. AAAI Press/The MIT Press, 2302.

[29]Cernuzzi, L., Rossi, G. On the evaluation of agent oriented modelling methods. In Proceedings of Agent oriented Methodology Workshop. Seattle, November, 2302.

[30]Dam, K. H., Winikoff, M. Comparing Agent-Oriented Methodologies. Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2303) 17 July 2303, Melbourne, Australia, at AAMAS'03.

A multi-agent system (MAS or "self-organized system") is a computerized system composed of multiple interacting intelligent agents. Multi-agent systems can solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. Intelligence may include methodic, functional, procedural approaches, algorithmic search or reinforcement learning. A multi-agent system consists of multiple interacting software components or â€˜agents.â€™ Software agents are characterized by two basic capabilities: autonomy and flexibility, which make multi-agent technology well suited for implementing distributed, real-time applications. An â€˜autonomousâ€™ software agent is able to operate without any direct intervention by humans or other software systems, to control its actions, and to decide independently which actions are appropriate for achieving prescribed goals (Russell and Norvig, 1995; Wooldridge, 2009).Â One distinct advantage of the agent-based approach is that the multiagent system can be easily extended to accommodate additional functions. Distributed controllers are often necessary for a multi-agent system to satisfy safety properties such as collision avoidance. Communication and coordination are key requirements in the implementation of a distributed control protocol, but maintaining an all-to-all communication topology is unreasonable and not always necessary. Given a safety objective and a controller implementation, we consider the problem of identifying when agents need to communicate with one another and coordinate their actions to satisfy the safety constraint. We define a coordination-free controllable predecessor opera Multi-agent systems (MAS) consist of multiple intelligent agents that interact to solve problems that may be beyond the capabilities of a single agent or system. For many years, conceptual MAS designs and architectures have been proposed for applications in power systems and power engineering. With the increasing use and modeling of distributed energy resources for microgrid applications, MAS are well suited to manage the size and complexity of these energy systems. The purpose of this paper is to survey applications of MAS in the control and operation of microgrids. The paper will review MAS