An Agent-based Middleware for Adaptive Systems

Nauman A. Qureshi, Anna Perini
Software Engineering Research Group
Fondazione Bruno Kessler - IRST
Via Sommarive, 18, 38050 Trento, Italy
{qureshi, perini}@fbk.eu

Abstract

New generation distributed software systems are expected to be able to meet changing user needs and to manage the variability of the open environment they operate in. This motivates current research on developing adaptive software applications, which recognizes a prominent role to middleware. In this paper, we discuss requirements for an agent-based middleware that enables adaptation. We illustrate these requirements as a result of modelling an application scenario that will be supported by our middleware, using an agent-oriented methodology. Middleware is conceived as web of agents, to overcome the challenge of run-time adaptation in an open environment.

1. Introduction

New generation distributed software systems are expected to be able to autonomously integrate with each other in various ways, while operating in an open and heterogeneous environment. While currently, distributed software systems are mainly subject to maintenance by human administrators, increasing the cost of maintenance, users of new generation systems require quick and effortless solutions to configure or align them to their current needs.

It is envisioned that these systems could manage themselves in the future without human intervention, realizing the so called autonomic computing paradigm, which is inspired from biological system properties [6, 10]. Autonomic software is characterized in terms of a set of properties, namely *self-configuration*, *self-optimization*, *self-healing*, *self-protection*, collectively proposed as *self-managing* or *self-** properties. If we envisage systems as *self-managing* then a system which can modifies itself satisfying the above mentioned qualities are said to be *Self-adaptive*.

Many different definitions of *Self-adaptive* systems have been proposed so far. According to [1], computer-based system capable of recognizing that the environment with which it shares an interface has changed and then of modifying its own behavior for adapting to these changing conditions is called dynamically adaptive systems. *Self-adaptive* systems are able to adapt to changing user needs and resource requirements at run-time, according to [4]. Systems adapting themselves toward changing environment providing dependability and robustness without human intervention are called *Self-adaptive* systems in [5].

In this work, we adopt the following definition for *Self-adaptive* system:
A software system that, at run-time, either maintains its state or adopts a different behavior with respect to the initial one as a consequence of change in user needs and operating environment.
Here, we can assume from our definition, that a distributed software system may adapt by running a new configuration variant, in response to user and environmental changes. The main focus of our work is to define appropriate software engineering methods and techniques complemented by agent technology to develop *Self-adaptive* software.

The main idea is to motivate the use of externalized adaptation mechanism, following previous proposals in software architecture research [3], by introducing an agent-based middleware to develop adaptive systems. Middleware, in classical sense, is considered to be a software layer that operates between operating system and applications. To us *it is a software layer acting as a mediator between operating system and user applications as a web of interacting agents*. Having assigned it this role, middleware can provide a solution to the problem of variability in software system usage, that is when we have various users with different and diverse set of skills, and we have to understand their preferences in using a particular software system. In this paper we will focus on adaptivity as a property which enables software *Self-configurability*.

High-variability in user preferences with respect to changing environment may lead to complex system behaviors. Goal-oriented requirement engineering [14], has

proved to be suitable to specify the alternative designs for the software. On the other side, Multi-Agent Systems [15] offer complementary techniques to take this variability from design to implementation, by using an agent-oriented methodology [11, 7].

In this paper we introduce an example to help illustrating variability in software usage and how it can be addressed enabling software *Self-configuration*. Using a suitable agent-oriented methodology and goal modelling techniques helps us to understand the requirements for our intended agent-based middleware, as a solution to realize *Self-configuration*. The role of this middleware in providing externalized adaptation mechanism to the modelled system is further discussed.

The paper is organized as follows. Section 2 presents related work. Section 3 outlines our motivation for using agents. Section 4 introduces the example that we use for highlighting the requirements for a middleware. Section 5 describes the middleware we propose. We then conclude in Section 6 with some future work.

2. Related work

The middleware we present, inspires from several interesting work realized so far, which leverage Architecture based approaches, Component Based (Middleware), Multi-Agent Systems (MAS) and Requirements Engineering paradigms.

For example, requirement engineering approaches for *Self-adaptive* software are presented in [1, 8]. By leveraging the analysis of goal alternatives, goal-oriented requirements engineering [14] is helpful in capturing the user need variability and helps in modelling the system as a family of products [7]. Moreover, by using the goal-oriented analysis it is meaningful to map user's high level preferences into software agents, which may select goals to be achieved and switch from a behaviour to a most appropriate one, accordingly [11].

Architecture based approaches are motivated in [3] - Rainbow project[1], providing an architectural model, which introduces the concept of externalized adaptation mechanism and reuse of adaptation properties. The idea of separation of adaptation concerns from the application concerns is followed in [5] with the focus to use a middleware (QuA)[2] to manage *Self-adaptivity* by mirror-based reflection.

In [4], *MADAM*[3] middleware is used to reason about the architectural models generated at run-time. Moreover, it emphasizes the application's variants and their properties by addressing coarse grained and fine grained variability. MADAM's further extension is *MUSIC*[4] - an open source platform for building mobile applications supported by tools and middleware. The architecture based approaches are supported by component based (Middleware) and uses the concept of (monitoring-evaluating-adapting) adaptation loop, but the challenge lies in answering questions like what, why, when and how to perform adaptation.

Following [6], we believe that software agents can provide a useful paradigm and technology to answer these questions when realizing autonomic computing systems.

The agent paradigm has been used to define and implement distributed system architectures called Multi-Agent Systems (MAS) [15] and as a relevant abstraction for modelling distributed software system.

In [13] the Unity platform for autonomic computing is an example of how MAS can be used in practice for enabling *self-** properties. This architectural model adopts software agents implemented in Java. It is emphasized that an agent-oriented middleware in systems made of agents is a suitable choice for seamless integration both at model and infrastructure level [9]. This paper provides motivation for our work. More recently, [12] illustrates the suitability of agents as core artefact. to model system dynamics entailing a requirements driven approach to develop self organizing MAS by expressing adaptivity in them.

Main aim of our work, is to find a synergy between software engineering methods and agent technology, to model and develop adaptive systems. We exploit the idea of externalizing adaptation mechanism and propose the use of an agent-based middleware as a solution to run-time adaptation, taking into account variability in user and environmental changes.

3. Agents as Adaptation Metaphor

Software agent can be viewed as a social, autonomous, proactive and reactive computational entity situated in an environment, capable of performing autonomous actions on behalf of its user by presenting flexible behaviors to fulfill the goals for which it has been designed [15].

From the above definition agents can be used as *"Adaptation Metaphor"*. This supports the choice of an agent-based middleware to enable adaptivity. An agent in our proposed middleware acts as an adaptation mediator to support the notion of externalized adaptation and reuse.

This idea of agent-based middleware development also poses a challenge on how to define an appropriate design process[5], which not only supports the development, but also

[1] http://www.cs.cmu.edu/ able/research/rainbow/
[2] http://simula.no/research/networks/projects/QuA
[3] http://www.ist-music.eu/MUSIC/madam-project/madam
[4] http://www.ist-music.eu/
[5] A preliminary description on design process for developing *Self-adaptive* systems can be found in a Technical report no. FBK-IRST TR#080405, Titled as *Towards a Design Framework for Self-adaptive Systems: A Process View* by Nauman A. Qureshi and Anna Perini

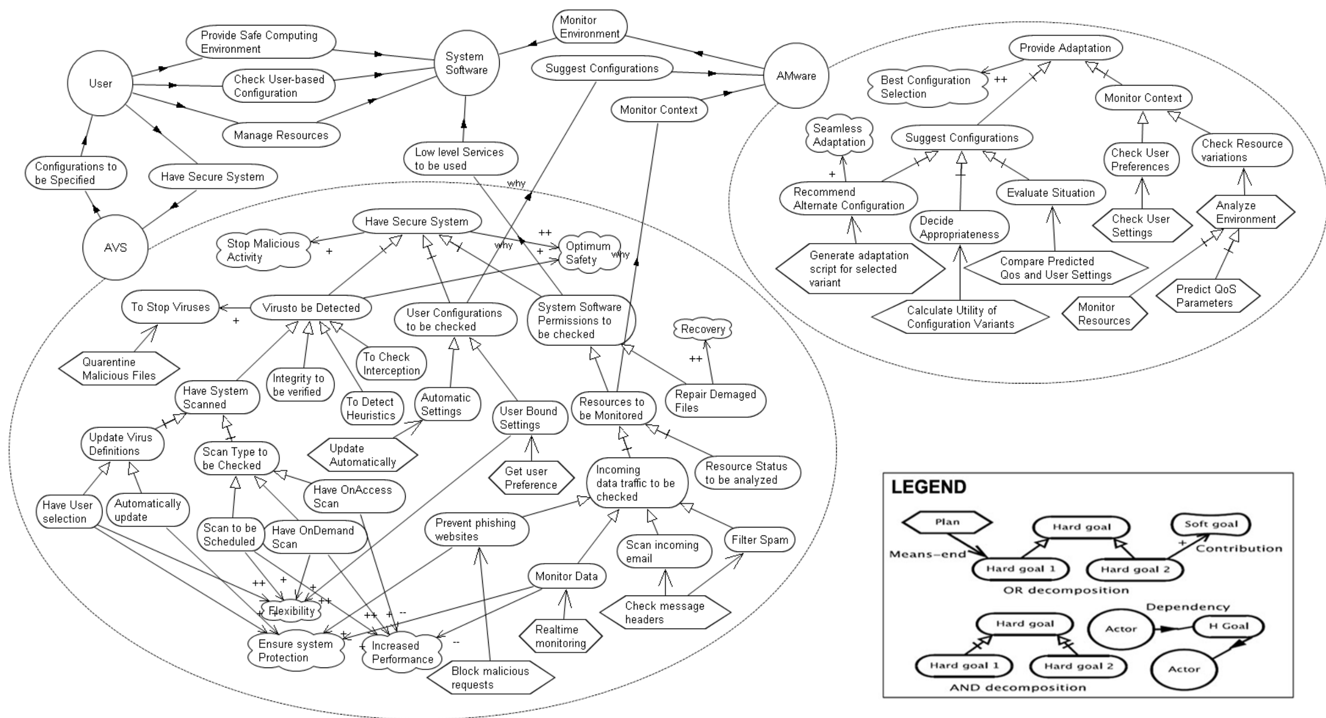

Figure 1. Tropos Late Requirements Model: Antivirus System and Adaptation Middleware

provides an extensible and maintainable mechanism to align requirements and environmental variability at run-time.

4. Illustrative Example

The scenario mentioned below will help in identifying potential requirements for detailing the middleware:

Scenario: *Mr. Chen is a corporate professional, who is working as a financial consultant in a firm. Chen is using a company's laptop machine for his office and home usage. He often travels to various places to attend business meetings. Chen has different profile settings for using his laptop machine from home, office and elsewhere. At home, his computer is also sometimes used by his wife and daughter to browse and check their emails. But at home he has normal internet connection from a company. How Mr. Chen can ensure that his important office data and his computer system is safe from the digital pests i.e. viruses, malwares, spywares, trojans and worm when his family uses his computer.*[6]

Before further analyzing the scenario, it is important to consider the quality of service (QoS) parameters, which an Antivirus system (AVS) and an Adaptation Middleware (AMWare) have to meet. For simplicity we consider only three of the following parameters:

1. **Bandwidth**: Either increase/decreased, it is required to update the antivirus components;
2. **System Performance**: Antivirus operations may be resource consuming, so this factor fluctuates letting the AMware to decide which among alternative configurations, better fit the current system performance needs;
3. **Flexibility**: Taking into account the user working context and preferences, AMware selects the best application configuration.

4.1 Goal-oriented Modelling

This section, illustrates a goal-oriented analysis of the scenario, which has been performed using the *Tropos* methodology [2]. Late requirements analysis in *Tropos* focuses mainly on the system-to-be (in our case: AMware, shown as an actor).

In Figure 1, the Antivirus System (AVS) is modelled as an actor (depicted as a circle), whom the user delegates a high level goal, namely Have Secure System. The AVS hard and soft goals are analyzed from the point of view of the AVS itself, in the diagram balloon (left). In particular, the delegated goal - Have Secure System, contributes positively to the high level soft goal - Optimum

[6] This scenario is not taken to model the security aspect rather to model high-variability in user needs and operating environmental changes which generates the need for externalized adaptation mechanism using middleware.

Safety. The goal Have Secure System is (AND) decomposed into three sub goals: Virus to be detected, User configurations to be checked, and System Software Permission to be checked. As antivirus programs operates, the AVS system must prevent the system from external threats and provide appropriate notification to the user and detects and repair damaged files. Virus to be detected is further decomposed into typical goals for an antivirus to detect viruses using (OR) decomposition (Have system scanned ... To check interception).

Moreover, the User configurations to be checked has further two alternatives: User Bound Setting and Automatic Settings with means-end plans to accomplish them. The System Software Permission to be checked goal is (OR) decomposed in two alternatives, namely the Resources to be Monitored and Repair Damaged Files sub goals, the latter giving (+) positive contribution to the soft goal Recovery. Here, goal Resources to be Monitored is further decomposed subsequently as shown in Figure 1.

The internal goals of the AMware represents adaptivity requirements. Goal delegation from the system actor (that is the AVS application) to AMware represent the externalization of basic adaptivity requirements of the AVS application itself (Suggest Configurations and Monitor Context goals). These delegated goals are internalized into the AMware and integrated with its top goal (Provide Adaptation) decomposition, which contributes positively to a high level soft goal Best Configuration Selection. Furthermore, Monitor Context is decomposed using (OR) decomposition with sub goals as Check User Preferences accomplished by a plan Check users Settings and goal Check Resource Variations with plan Analyze Environment. This Analyze Environment plan is (AND) decomposed by Monitor Resources and Predict QoS Parameters. The plan Predict QoS Parameters predicts the variations in the QoS parameters values as mentioned.

Simultaneously, Suggest Configurations is decomposed in three sub goals using (AND) decomposition, Evaluate Situation with the plan Compare Predicted QoS and User Settings, Decide Appropriateness with the plan Calculate Utility of Configuration Variants and Recommend Alternate Configuration with the plan Generate Adaptation Script for Selected Variant contributing positively to a soft goal Seamless Adaptation.

Here, AMware is responsible to monitor the context by predicting the QoS parameters, further evaluations and decision is made on the basis of comparison and calculation of of the configuration variant by using *Utility* functions (Calculate Utility of Configuration Variants plan). Utility functions provides, a recursive mechanism to reason about the alternatives, by determining the "Best" option. Lastly, AMware recommends the best configuration for AVS by generating an adaptation script to be enforced by the AVS at run-time.

So far, we have analyzed the goal delegation between AVS and AMware and decomposition of their goals, contributing to high level soft goals. In goal models alternate goal decomposition helps the architect to better analyze the variability by addressing the *WHY* for adaptation. This leads to high-variability design for run-time adaptation [11]. This modelling experience helped us to derive some requirements which are seminal for our intended agent-based middleware.

4.2. Identifying Requirements for Middleware

To this stage, we use goal models, which provides a mechanism to evaluates alternate ways for examining the user's requirements and sources of variability. Revisiting the existing architecture based (Middleware) approaches [3, 4, 5] in the light of this modelling experience, we have derived the following important questions that our intended middleware should allow to answer:

Q1: What adaptation concerns are to be externalized?

Q2: What to monitor in application and in operating environment?

Q3: What adaptation properties can be considered to reuse?

Q4: What level of control an agent can exercise while monitoring and deciding as an external entity?

By decomposing the system in the form of agents - in our case decomposing the AMware into a web of agents - we end up with the layered view depicted in Figure 2.

5. Agent-Based Middleware

In this section we sketche the AMware middleware, as a suitable option for providing externalized adaptation mechanism and enabling the seamless mapping of user requirements and environmental variability to the run-time needs. To detail this, we can see that the typical middleware services spans from traditional brokerage services to recently context-aware applications. Here, we present our middleware architecture as web of agents as shown in Figure 3.We start by defining the agents role to address the (HOW) and (WHEN) question about adaptation.

In Figure 2, we have decomposed the AMware into three layers as web of agents. The first layer is a composition of four main agents namely *Context Monitor Agent*, *Configuration Selector Agent*, *Enforcer Agent* and *Visualizer Agent* simulating the adaptation loop (Monitor-Evaluate-Adapt). Second layer composes wrapper agents which encapsulate

some monitoring components, providing information about changes occurring from or to the operating environment. Finally, the third layer is the agent platform, providing white and yellow page services to the upper layers. In this paper, we will discuss the roles and responsibilities of all the layered agents by referring to the requirements identified earlier in section 4. Layer 1 agents coordinates with each others and with layer 2, to provide adaptation mechanism. Layer 2 agents, provides wrapper agents [9] to mask low level information components - providing event details for higher agents to evaluate. Layer 3 agents provides infrastructural services to the other two layers. We now further see the individual roles of these agents.

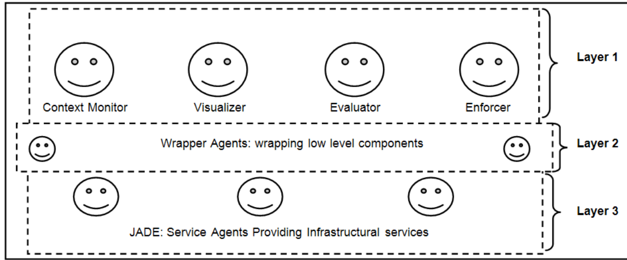

Figure 2. Layered View of Middleware

5.1. Context Monitor Agent

The main aim of the Context Monitor Agent is to observe the changes in the environment and user needs as configuration settings. For example as shown in Figure 1, this agent observes the environmental changes by predicting the QoS parameters and by exploiting the use of wrapper agents as shown in Figure 3. The wrapper agents are in fact low level system components, which provides event level information about the operating environment to the context monitor agent. This partially fulfill the requirements (Q2, Q4).

This agent starts monitoring the current context of the environment by knowing the current state of the system and application settings using wrapper agents. For example, the system state can be the current bandwidth, memory usage, processor performance etc. also user profile settings having access rights and user application settings etc. To get this information at run-time, this agent properly notifies the configuration selector agent by sending an appropriate messages about the evaluation and decision about the latest variations. It keeps record for the latest and run-time variations to observe some patterns of events.

5.2. Configuration Selector Agent

The goal of this agent is to plan and evaluate. The agent evaluates by comparing the predicted information provided by the context monitor agent as shown in Figure 3. To calculate the utilities of the selected configurations for the application system as shown in Figure 1. The process of calculating the utility is recursive, hence this agent judges the selected configuration variants by assigning weights in a recursive manner.

This mechanism helps in reasoning about the alternatives in more deterministic way. In the end, the configuration variant having the highest weight will be selected. Here, our focus is to map and evaluate the user preferences by re-evaluating the configurations of the application system.

This agent starts knowing the variable configurations that can be selected under certain circumstances i.e. Configuration selector selects the configurations based on the information sent by the context monitor agent and also referencing to the environment. Exploiting this information by evaluating it and plans to inform the enforcer agent to generate the adaptation script thus satisfying the requirements (Q1, Q4).

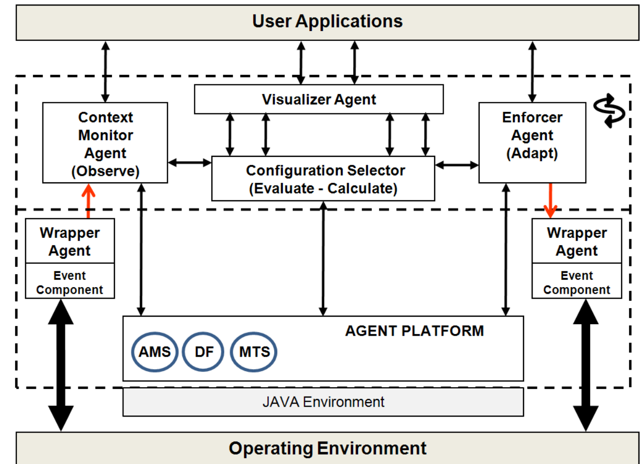

Figure 3. Agent-based Middleware Architecture

5.3. Enforcer Agent

Enforcer Agent provides seamless integration of the selected and evaluated configuration variants. This agent exploits the information provided by the configuration selector agent. The main aim includes continued service delivery to the user by generating a adaptation script as shown in Figure 1. Based on the configuration variant selected for the system to adapt. Enforcement mechanism includes twofold process, one is the generation of automatic script and second is to enforce or run the adaptation script during run-time without compromising the system operations thus partially fulfilling the requirements (Q3, Q4).

This agent implements the current settings/configurations selected by the configuration selector i.e. In order to enforce the latest configurations the enforcer agent uses wrapper agent as shown in the Figure 3 to enforce the generated adaptation script. This assures that the system is optimized by shifting to the best selected configurations. It also ensures the satisfaction of the configurations selection and helps user with proper messages by publishing notification messages and writing log about the enforcement process details.

5.4. Visualizer Agent

Visualizer agent holds the minimal role as compared to other agents, so far the role of this agent is to display the information for the user to visualize the adaptation process, notifications and other information generated during the whole working of the middleware. This agent gets the information mainly from the configuration selector agent but indirectly from other agents as well.

6. Conclusion & Future Work

In this paper, we have discussed an agent-based middleware which meets *Self-adaptivity* requirements, exemplified in an illustrative example that we modelled according to a goal-oriented approach. We claim that using agents as middleware allows to address reuse and scalability issues. This provides a way to have many other agents with focused roles, which can be introduced at needs. The middleware we have introduced can be generalized to provide an externalized adaptation mechanism to multiple applications.

As our future research, we will be considering much finer details about our middleware for developing it and evaluating with much realistic examples. This will strengthen our intended approach towards *Self-adaptive* systems and will enable us to better exploit the design process for mapping requirements and environmental variability at run-time. As a further extension users can also be facilitated to specify policies for adaptation at high-level using our visualizer agent. The user policies can be enforced automatically, hence the system will be driven by automated policies rather then by humans relieving them from the decision-making processes related to adaptation or control loop tasks.

References

[1] D. M. Berry, B. H. Cheng, and J. Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ05)*, pages 95–100, 2005.

[2] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[3] S.-W. Cheng, A.-C. Huang, D. Garlan, B. R. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. In *ICAC*, pages 276–277. IEEE Computer Society, 2004.

[4] J. Floch, S. O. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjørven. Using architecture models for runtime adaptability. *IEEE Software*, 23(2):62–70, 2006.

[5] E. Gjørven, F. Eliassen, K. Lund, V. S. W. Eide, and R. Staehli. Self-adaptive systems: A middleware managed approach. In A. Keller and J.-P. Martin-Flatin, editors, *Self-Man*, volume 3996 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2006.

[6] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[7] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. On goal-based variability acquisition and analysis. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE'06)*. IEEE Computer Society, September 2006.

[8] M. Morandini, L. Penserini, and A. Perini. Towards goal-oriented development of self-adaptive systems. In *Proceedings of (SEAMS '08)Workshop on Software engineering for adaptive and self-managing systems*, pages 9–16. ACM, 2008.

[9] A. Omicini and G. Rimassa. Towards seamless agent middleware. In *WETICE*, pages 417–422. IEEE Computer Society, 2004.

[10] M. Parashar and S. Hariri. Autonomic computing: An overview. In J.-P. Banâtre, P. Fradet, J.-L. Giavitto, and O. Michel, editors, *UPP*, volume 3566 of *Lecture Notes in Computer Science*, pages 257–269. Springer, 2004.

[11] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. High variability design for software agents: Extending tropos. *TAAS*, 2(4), 2007.

[12] J. Sudeikat and W. Renz. Toward requirements engineering for self - organizing multi- agent systems. In *SASO*, pages 299–302. IEEE Computer Society, 2007.

[13] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White. A multi-agent systems approach to autonomic computing. In *AAMAS*, pages 464–471. IEEE Computer Society, 2004.

[14] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *5th IEEE International Symposium on Requirements Engineering (RE)*, page 249. IEEE Computer Society, 2001.

[15] M. Woolridge. *Multi-Agent Systems: An Introduction*. John Wiley & Sons (Chichester, England), 2001.