Modelling a Smart Music Player with a Hybrid Agent-Oriented Methodology

Yuxiu Luo Leon Sterling Kuldar Taveter
Department of Computer Science and Software Engineering
University of Melbourne, Vic. 3010
{yxluo, leon, kuldar}@csse.unimelb.edu.au

Abstract

This paper presents experience with using a hybrid agent-oriented software engineering methodology for designing an agent-based software system. We elicit and analyse system requirements using ROADMAP, with associated tool REBEL. We develop initial system design using Prometheus with associated tool PDT. Applying the combination of the methodologies facilitates understanding of system requirements by non-technical clients. A key feature of our work is its handling of quality requirements, which are identified as an essential asset for analysing design alternatives. We contribute to the current state of art by demonstrating the application of a hybrid methodology with a case study of a smart music player. We created concrete artefacts to represent models and found and clarified conflicting concepts between the two methodologies when we developed the case study. Suggestions are proposed to facilitate goal elicitation and verification.

1. Introduction

Agent-oriented software engineering (AOSE) is a recent contribution to the field of software engineering. It has several benefits compared to existing approaches to developing large-scale complex systems [7, 17]. In particular, agent-oriented software engineering has the ability to represent high-level abstractions of active entities as agents [15], and the ability to address crucial requirements engineering concerns in building complex systems, such as functionality, quality, and process.

Several agent-oriented software engineering methodologies have been proposed including Gaia [16], MESSAGE [2], ROADMAP [9], RAP/AOR [13], Tropos [1], Prometheus [11], and MaSE [5]. This paper shows how features of two methodologies, ROADMAP and Prometheus, can be combined into a hybrid methodology, and used for a substantive example.

Initial system design is included here because analysis of requirements will lead to a design proposal, and analysis of the design will show the need for further requirements [20].

We believe that there are advantages in using a hybrid methodology. In our case study, effective requirement elicitation and analysis was supported by ROADMAP, and effective initial system design was supported by Prometheus. We used a combination of tools, Rebel and PDT, to facilitate the system development process. Both tools [18, 19] are easy to use based on our experiences

The paper is organized as follows. Section 2 introduces the techniques XPod [6] and Push!Music [8] which we are reusing in our smart music, followed by an overview of our desired system. In section 3, we analyse the system using ROADMAP. In section 4, we provide initial system design using Prometheus. We present examples of the artefacts as we proceed to make the experience concrete. Section 5 of the paper discusses significant aspects of the case study, while Section 6 concludes.

2. Smart Music Player

The objective is to design a multi-agent system controlling a Smart Music Player (SMP) to improve listeners' experience by automating most interactions between music player and listeners, and by providing flexible system control. We focus on the main functionalities related to playing music. Additional possible functionalities, such as showing photos, receiving radio, and showing video, are out of scope.

A PC is assumed available to back up songs from the music player. The BodyMedia SensorWare sensor package[1] is assumed to gather physiological data. A voice recorder is assumed to record audio files. The listener informs the player about music preferences during initial training. Listeners also train regarding voice commands. The music player is in a smart home environment where signals can be sent between phone, SMP and PC[2]. The ring tone of the phone is recorded

[1] It is used to predict emotions and activities of the listener. The BodyMedia device uses a series of sensors to measure the rate of body movement, acceleration, and body heat from the listener and wirelessly transmits the data obtained to the server (SMP in our case).
[2] This requirement comes partly from an ARC research project, where music can be muted during a phone call.

1090-705X/07 $25.00 © 2007 IEEE
DOI 10.1109/RE.2007.43

before use. Online music sites are setup before a player downloads music from the net. Note that the project proceeded before the announcement of the iPhone.

We base our design on XPod technique [6] and Push!Music technique [8] to ensure feasibility. Our system provides functions including music playing, song selection, volume control, track control, state checking, and Music library management. Details are available from the authors.

3. Requirements analysis

We chose ROADMAP [9] for requirements analysis because it represents requirements at different abstraction levels using goals and roles. During early analysis, requirements are captured accurately at a high level by goal and role diagrams without specifying all details. During later analysis, relevant functional and quality requirements (QRs) are described in detail by refining goal and role diagrams at a lower level. Roles are captured by the role model. A role can be played by a human or software agent. The human perspective provided by using roles helps to identify system goals and important features such as quality goals (QGs) to meet stakeholders' intentions especially when working with non-technical people. A scenario complements goals and new goals may be revealed when specifying scenarios. Scenarios also help to verify goals.

3.1. Requirements elicitation

First, we suggest a few information sources from where goals can be elicited.

Customers: Their business objectives ultimately determine system goals. Extra information that may reveal system goals based on organization-specific issues (e.g. the availability of resources, standards or any other constraints) are also provided by them.

Marketing people: They know what the market really needs and therefore help to decide the most desired requirements when stakeholders mix what they want with what they need. Besides, they offer information on the history, current situation, and future trends of the market that tell from which problems have systems of this type suffered. This information reveals the system's additional functional goals and QGs. Information about the current market is especially important for identifying QGs. Take security as an example, because attackers attack systems that are easier to break in, we need to build a system that is harder to break into than a competitor's system. To achieve this, we need information about similar products on the market. Furthermore, experienced marketing people can suggest what the system should look like.

Development team and/or domain expert: they have good knowledge about the application and development environment. They know hardware requirements and may propose new requirements due to development constraints (e.g. operating system(s), the platform(s), the component type, and component interface used)

End user: user's thoughts greatly affect the QGs (e.g. response time) and other constraints (e.g., cultural factors that influence UI style). Some abstract or vague system requirements may become telling and clearer if they are described from the user's point of view.

Available documents: legacy system documents, E-policies, laws, and standards all reveal functional goals and QGs.

The next step is to collect information from the above information sources.

We recommend collecting requirements via a small number of requirement workshops instead of by many single interviews. Stakeholders of the system (i.e., customers, marketing people, development team, and/or domain experts) are brought together in a group meeting to save time and collect more complete and more meaningful requirements because the stakeholders have a better understanding of the requirements from each other's points of view.

For end users, we suggest face-to-face interviews to help to clarify specific requirements because one can elicit a lot of information quickly from a single person and people will tell you things privately that they would not tell publicly. If listeners are dispersed, electronic interviews can be held instead of face-to-face interviews. However, since limited information can be conveyed electronically, face-to-face interviews should be preferred over electronic ones.

For the available documents, the method proposed in [14] can be used to identify goals from documents, interview transcripts, etc. by searching for intentional keywords.

Paper [3] presents a way to specify roles for a multi-agent system. Role details are captured in the role model, which is described in Section 3.2.

3.2. Goal models

In a goal model, goal and role diagrams are produced to capture and analyse system requirements.

3.3.1 Building goal models. After the goals and roles have been identified, we use *goal and role diagrams* to represent requirements in a clear manner that is easy for stakeholders to understand for the purpose of validation. Roles and QGs are attached to functional goals. QGs constrain how a goal should be achieved. They reflect intangible requirements of a system, such as privacy, security, and performance. Roles represent those responsible for achieving goals. Figure 1 below shows the notations of the Rebel tool, which we have used for creating goal models.

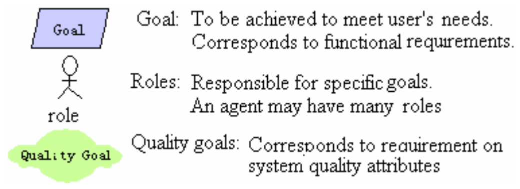
Goal
Goal: To be achieved to meet user's needs. Corresponds to functional requirements.
role
Roles: Responsible for specific goals. An agent may have many roles
Quality Goal
Quality goals: Corresponds to requirement on system quality attributes

Figure 1. Requirements analysis notation

Figure 2 is the goal and role diagram for a SMP system, which delivers a clear and easily understandable picture of the overall system. The overall goal is to *manage playing music*. The *music player manager* role is responsible for achieving this goal. The QG *easy to use* indicates that it should be easy to manage playing music. The goal *manage playing music* can be achieved via several sub-goals: *handle listener request*, *determine setting*, *monitor environment*, *play music*, *run message,* and *manage music library*. These sub-goals have roles and QGs attached.

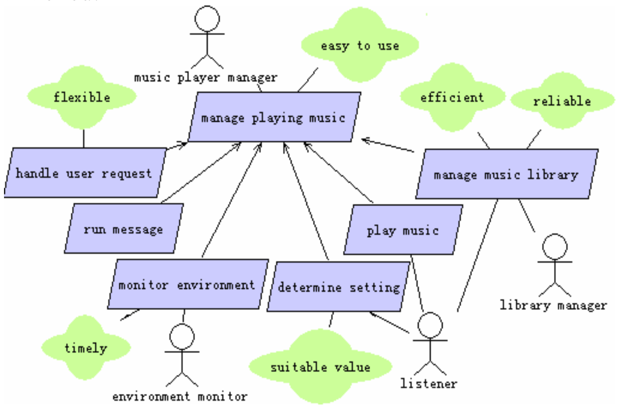

Figure 2. Goal and role diagram for SMP system

System goals are further explored during requirements analysis. Figure 3 shows the sub-goals of the goal to *manage music library* goal.

Some research (e.g. [1], [4], [10]) advocates more information when analysing goals such as AND-OR relationships between functional goals, or positive/negative influences on QGs. However, we believe that it is important to separate requirements analysis from design.

As is shown in Figures 2 and 3, goals are presented in a loose hierarchy. The high abstraction level and simplicity of the goal hierarchy hide complexity from clients. This is very helpful for communication between system developers and non-technical clients during requirements elicitation. Further, the multiple abstraction levels provided by the hierarchy can be used to refine the system during requirements analysis. ROADMAP's simple concepts lead to flexible usage of the Rebel requirements engineering tool, which can significantly shorten the learning curve for developers.

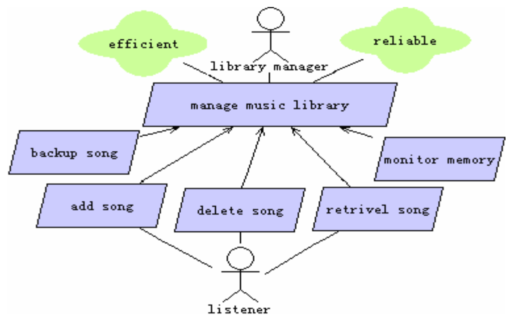

Figure 3. Goal model for *manage music library*

3.3.2 Quality goals. QGs can also have sub-goals. Sub-goals can be a set of sub QGs and/or a set of functional sub goals. Note that the achievement of QGs associated with sub-goals does not necessarily ensure the achievement of the QG associated with the top goal. In Figure 2, for example, we cannot guarantee that the system is easy to use if the listener's requests are handled flexibly; there is timely monitoring of the environment, suitable values of settings are determined, and music library management is reliable and efficient. In fact, QGs may conflict. For instance, if the ways for listeners to make requests are too flexible, the system may not be easy to use, because of too many options available, which results in confusion.

3. 3. Role models

A role has a coherent set of responsibilities specifying what the actor who plays the role is expected to do within the organization. Each role is responsible for achieving, or helping to achieve or maintain specific system goals. The role model captures details of the roles that are identified in the goal model. *Description* gives a brief statement about the role. *Responsibilities* include the essential requirements on the role to achieve its associated goals. *Constraints* document the constraints associated with fulfilling the responsibilities. They can also constrain how a QG is to be achieved or the knowledge is to be accessed. A role in our case study can be performed by a human or a software agent.

Table 1 shows responsibilities and constraints of the environment monitor role. An environment monitor must detect ring tones, monitor ambient decibel levels, and accept hang-up signals. It analyses the listener's activities and collects relevant data when the listener is nearby. It has quality constraints on performance requirements.

Table 1. Environment monitor role model

Role name	Environment monitor
Description	Monitor the environment, recognize and process different signals to control volume and gather information for the play list producer.
Responsibilities	• Detect ring tones and receive "call finished" signals • Detect the decibel level of the surrounding environment • Detect if the listener is nearby. • Monitor body movements, acceleration and body heat of the listener and transmit this information to the play list producer • Update BodyMedia physiological data
Constraints	Detect and process signal within N seconds. N depends on the performance requirements.

Role models not only provide information about system roles but also describe listener expectations (see Table 1). This is an advantage over other methodologies. The lists of responsibilities and constraints facilitate giving feedback. They represent important details about envisaged system behaviour. Role models are thus helpful for deriving test cases and for initiating lively discussions with clients about responsibilities and constraints of roles when eliciting requirements.

3. 4. Scenarios

A few scenarios can be created to complement goal and role models with the descriptions of system behaviour. Scenarios describe system behaviour in terms of system goals, system operations, and impact on external data. Each scenario consists of several steps. Each step is one of the five types: Goal (G), Action (A), Percept (P), sub-Scenario(S), or Other (O). For each step, data may be read and/or produced (R: read; W: write). Actions represent how the system acts on the external environment and percepts represent events/stimuli in the outside world to which the system reacts. Because a scenario captures only one particular sequence of steps, it can be useful to indicate small variations with a brief description. A major variation should be presented as a separate scenario.

Table 2. Song selection scenario

Scenario: song selection

Description: smart music player automatically selects and play the songs best suited to user's emotion and the current activity

Trigger: user request to play music

Steps:

#	type	Description	Data
1	O	Play list is requested	
2	P	User is nearby	
3	O	Determine current states	user metadata (R,W)
4	G	Setup playlist	song playing history(R,W)
5	O	Send play list	Music library(R)

Variation1: user is not nearby

Description: At step 2, user is not near by, then smart music player produce song list by user preference

Steps:

#	type	Description	Data
2	P	User not nearby	
3	G	setup play list based on user preference	song playing history(R,W)

Table 2 represents an example scenario of song selection. It describes how the SMP automatically selects and plays songs best suited to the listener's emotions and current activities. Table 2 shows two variations for step 2. One of them occurs when the listener is not near the SMP. In this case, the SMP produces a song list by listener preferences. Another scenario occurs when the listener selects songs by himself or herself.

3. 5. Requirements validation and verification

Goal and role diagrams can represent system goals at high abstraction level in a clear manner that is easy for stakeholders to understand. This facilitates validation with stakeholders, especially with non-technical stakeholders.

Requirements need to be verified after being validated by stakeholders. We believe that goals need to be verified based on both scenarios and roles.

Each goal needs to be achieved by at least one scenario; otherwise the goal may not be feasible or necessary. A scenario must realize at least one goal, otherwise there may exist goals that are missed out from goal and role diagrams.

A role should be assigned to at least one goal. If no goal is associated with a role, then the role is not needed. Alternatively, it could be an external role that interacts with the system. If no role is assigned to a goal, then the goal may be unfeasible and should be removed from the goal model. Alternatively, a role may have been missed when identifying goals and roles.

After verifying the goals, roles and scenarios are included in the verification, where we can apply the methods mentioned in [14] to detect inconsistencies between scenarios and goals. If a goal violation has been found, the goal model needs to be modified; new goals or roles may be needed to prevent the violation.

4. Initial design

Prometheus [11] was chosen for design because of its effectiveness in assisting developers to design, document, and build multi-agent systems. During architectural design, the models produced by requirements analysis are used to determine what agents should exist and how they should interact. Prometheus provides two mechanisms to analyse potential groupings of roles to decide agents: data coupling and agent-role coupling. Here we focus on clarifying the conflicting role concepts between ROADMAP and Prometheus instead of describing the details about system design.

Figure 4 shows how roles in the SMP system have been mapped into agents. Agent-role coupling was easily understandable for non-technical clients because of its analogy to humans and their roles and responsibilities within an organization. It also enabled checking role assignments against fulfilment of goals.

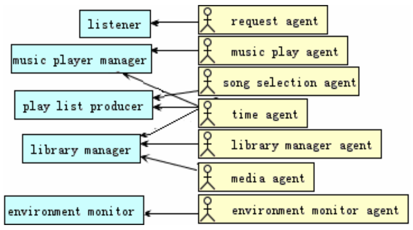

Figure 2. Agent-role coupling diagram

In our case study, some roles have been mapped to several agents instead of being mapped to one agent as is described in [11]. In Prometheus, the concept of role is defined at a low level. Roles are functionalities; thus, when using Prometheus in design, to group roles into an agent means to group functionalities to form an agent. However, in ROADMAP, the system is analysed at a high level to hide the complexity by using goal diagrams. The system is analysed in a top-down manner. First, we define goals and then identify roles that are needed for achieving the goals. After that, we further define each role's responsibilities using a role model. To fulfil a responsibility, a role should include several functionalities. A part of the responsibility of different roles can be achieved using the same or similar functionalities, whereas an agent should encompass sets of similar functionalities. In this sense, in our hybrid methodology, several agents work together to achieve the responsibilities of one role and one agent can play several roles; while several roles can be mapped to one agent in Prometheus. Figure 4 shows the responsibility of role *play list producer* is fulfilled by *time agent* and *song selection agent*. *Time agent* services 3 roles: *music player manager, play list producer* and *library manager*. The difference of agent mapping between ROADMAP and Prometheus is due to the different concept of role. Our hybrid methodology analyses system goals and roles in a top-down manner, which we believe is easier for non-technical clients to understand.

5. Conclusions and future work

We have described a hybrid agent-oriented software engineering methodology via a case study of a smart music player.

Several information sources and corresponding elicitation techniques have been suggested for eliciting goals. ROADMAP provides an easy to represent system expectation. The high abstract level of goal models hides complexity from non-technical clients. Goal models are refined into a more concrete level for the purpose of sound requirement analysis. The hierarchical structure of goal diagrams enables the requirements to be captured flexibly. During the elicitation phase, a high abstraction level hides complexity from non-technical clients, so that they can understand what requirements have been identified and decide if the system to be developed meets their needs. During the later analysis phase, goals can be refined in more detail. The hierarchy also allows scaling of requirements. One project in our lab had an understandable requirements document of over ninety pages which was based on goal diagrams [12].

Role models capture details about roles similarly to how clients describe roles in their organization, enabling quick feedback from the clients. Role models also show user expectations, which are useful for developing user manuals and making design decisions.

System behaviour is captured by scenarios. They are useful for integration of all the models that are produced at all design levels. Scenarios are similar to the use case scenarios in UML. Concepts used in scenarios, such as percept, goal, and action, are very simple. According to our experience, developers do not have difficulties to produce scenarios.

Finally, rules are proposed for goal verification. And we show how roles in the SMP system have been mapped into agents when applying Prometheus.

In summary, ROADMAP represents agent concepts that are easy for both non-technical clients and developers to understand because the concepts used – role, goal, and quality goal – are close to how humans understand the world. For client, not too many details are represented during requirement elicitation process. For developers, the tools are easy to use with little effort needed to learn how to model the new aspects of requirements even when they don't have agent-oriented background. We have significant experience through teaching over the past three years to support this claim.

Based on our experience, ROADMAP provides an organizational view of computing that remedies the surging complexity in large-scale industry-strength systems with strong intelligence requirements. It facilitates communication between developers and clients. Requirement elicitation and analysis process is quite effective. Goal model, role model and scenarios clearly captures information needed in system design to drive agents, interactions among agents, data shared and their interfaces to environment in terms of percepts, actions and external data. Goal are associated with particular roles and quality goals, constraining the design alternatives and helping developers to make trade-offs. We have omitted the remaining diagrams due to space limitations. They illustrate the comprehensiveness of the design and are available from the authors.

The concrete nature of Prometheus, and its detailed models and process, help developers to decide which agents should exist within the system and what functionalities they should have. We keep goal models and scenarios simple to avoid "over-analysing" and "over-designing" the system.

Overall, the combination worked well, except that we needed to consider some differences in the role concept when defining the agents. Agent concepts facilitated the understanding of software design. QRs, which are important for all kinds of applications, were captured using QGs. In addition, since agent concepts are helpful for analysing and designing all kinds of software systems, and our methodology has been shown to work well by this case study, we believe our methodology should not be limited to the developing of agent-based systems.

We have also discovered other interesting directions for future work. One of them is that QRs should be considered at the beginning of a Software Development Lifecycle (SDLC). They should be tightly built into the software system to be built. Therefore, we are in the process of developing a general framework for unifying QR concepts and presenting their inter-relationships using agent concepts. Based on the framework, a methodology can be proposed to logically model, analyse, design, verify, and measure QGs (i.e., QRs) in a particular context. It guides developers to analyse QRs from early stage of SDLC, make proper trade-offs (when conflicts exist among QGs) and systematically add QRs into the system through each phase of SDLC. The Tropos methodology [1] is one of the most widely published approaches trying to attack this problem. However, the diagrams produced by Tropos easily become complex and there is a limited tool support so far. Thus, integrating QGs into a multi-agent system in a good manner is one of the problems to be solved in the future. In addition, the mapping of QGs from analysis to architectural design should be explored.

6. References

[1] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia and John Mylopoulos, "A knowledge level software engineering methodology for agent oriented programming", In *Proc. of the Fifth International Conference on Autonomous Agents*, 2001. pp. 648-655.

[2] Giovanni Caire, Francisco Leal, Paulo Chainho, Richard Evans, Francisco Garijo, Jorge Gomez, Juan Pavon, Paul Kearney, Jamie Stark, and Phillipe Massonet, "Agent oriented analysis using MESSAGE/UML", in Michael Wooldridge, Paolo Ciancarini, and Gerhard Weiss, editors, *Second International Workshop in Agent-Oriented* Software Engineering , 2001, pp. 101-108.

[3] K. Chan and L. Sterling, "Specifying Roles within Agent-Orientated Software Engineering." In *Proc. of the Tenth Asia-Pacific Software Engineering Conference*, Chiang Mai, Thailand, December 2003, pp. 390-395.

[4] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite, "Nonfunctional Requirements: From Elicitation to Conceptual Models". *IEEE Trans. Software Eng. 2004, 30*(5): 328-350.

[5] Scott A. Deloach, Mark F. Wood and Clint H. Sparkman, "Multiagent Systems Engineering*", International Journal of Software Engineering and Knowledge Engineering*, 2001, 11(3): 231-25.

[6] S. Dornbush, K. Fisher, K. McKay, A. Prikhodko and Z. Segall, "XPOD – A Human Activity and Emotion Aware Mobile Music Player", *Proc. of the International Conference on Mobile Technology, Applications and Systems*, 2005.

[7] Luiz Márcio Cysneiros Filho, Vera Werneck, Juliana Amaral, Eric SK Yu, "Agent/goal Orientation versus Object Orientation for Requirements Engineering: A Practical Evaluation Using an Exemplar." In *Proc. of VIII Workshop in Requirements Engineering,* 2005, pp. 123-134.

[8] Mattias Jacobsson, Mattias Rost, Maria Hiansson and Lars Erik Holmquist, "Push!Music: Intelligent Music Sharing on Mobile Devices", In *Adjunct Proc. of UbiComp*, 2005, Tokyo, Japan. Demonstration.

[9] Juan, T., Pearce, A., and Sterling, L, "ROADMAP: Extending the Gaia Methodology for Complex Open Systems", in *Proc. 1st Intl. Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002, pp. 3-10.

[10] J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu, "Exploring Alternatives during Requirements Analysis". *IEEE Software*, 2001. vol. 18, pp. 92-96.

[11] Padgham, L., Winikoff, M. *Developing Intelligent Agent Systems*. John Wiley & Sons, 2004.

[12] Sterling, L., Taveter, K. and Daedalus Team, "Building Agent-Based Appliances with Complementary Methodologies." in *Proc. of the Joint Conference on Knowledge-Based Software Engineering 2006*, Estonia.

[13] Taveter, K. and Wagner, G. "Towards Radical Agent-Oriented Software Engineering Processes Based on AOR Modelling." In: Henderson-Sellers, B., Giorgini, P. (eds.), *Agent-Oriented Methodologies*. Idea Group Publishing, 2005. pp. 227-316.

[14] A. van Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective". Invited Keynote Paper, In *Proc. of 22nd International Conference on Software Engineering,* ACM Press, 2000, pp. 5-19.

[15] Michael Wooldridge, "Intelligent Agents", book chapter in *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Edited by Gerhard Weiss. pp. 1-77.

[16] Michael Wooldridge, Nicholars R. Jennings, and David Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", J*ournal for Autonomous Agents and Multi-Agent Systems, 2000,* 3(3):285-312.

[17] Eric S. K. Yu, "Why Agent-Oriented Requirements Engineering", In *Proc. of 3rd International Workshop on Requirements Engineering: Foundations for Software Quality*, June 16-17, 1997.

[18] PDT http://www.cs.rmit.edu.au/agents/pdt/, last visited: 13/07/2007.

[19] REBEL http://www.cs.mu.oz.au/agentlab/, last visited: 13/07/2007.

[20] Nuseibeh. B. A," Weaving Together Requirements and Architectures." *IEEE Computer*, 2001, 34 (3):115-117.