Agent/goal Orientation versus Object Orientation for Requirements Engineering: A Practical Evaluation Using an Exemplar

Luiz Marcio Cysneiros[1], Vera Werneck[2], Juliana Amaral[1] and Eric Yu[3]
[1]*Dept. of Math. & Stat.- Information Technology Program*
York University – Toronto – Canada
cysneiros@mathstat.yorku.ca
[2]*Informatic and Computer Science Department*
Rio de Janeiro State University (UERJ) – Brazil
vera@ime.uerj.br
[3]*Faculty of Information Studies – University of Toronto - Canada*
yu@fis.utoronto.ca

Abstract. There are many different approaches to understand and model system requirements. However, systems today tend to be increasingly complex. Agent- and goal-oriented paradigms have been proposed as an alternative to object orientation to cope with these demands. Although it may be intuitive that object-oriented methods could not readily deal with issues such as autonomy, proactiviness and sociality, it is not yet clear to what extent this may be true. Thus, it is necessary to show not only where object orientation might fail, but also where agent/goal orientation still needs improvement. A practical approach that uses a well defined and complex problem producing specifications using agent/goal orientation and object orientation could guide us to understand better the strengths and weaknesses of each approach. That is the main goal of this paper. We use an exemplar proposed in 2001 by Yu and Cysneiros [1] to evaluate both agent/goal orientation and object orientation. For the agent/goal approach we use i*/Tropos, while for object orientation we use UML/RUP. Three teams applied both approaches to the exemplar, producing all the necessary models and answering the evaluation questions provided in the exemplar. We comment in detail on the key findings.

Keywords: Agent-Oriented, Object-Oriented, Systems Development Methodology, Requirements Engineering, Evaluation.

1. Introduction

Software is increasingly becoming part of everyday life; as a consequence, traditional conceptions of software are being extended to deal with more complex problems. In earlier conceptions, software information systems are conceived of as automating routine processes, as maintaining data in databases, or as reactive and interacting objects. However, to deal with today's complex problems, the requirements engineering community has pointed out the need to go beyond such conceptions. Works like GBRAM [2], KAOS [3] and Tropos [4] have been pointing out that it is necessary to deal with broader aspects in order to be able to understand and model requirements for these systems. The emerging agent/goal-oriented paradigm conceives software as being proactive and exhibiting autonomy and sociality. This orientation parallels the shift in applications towards open networked

environments, both in terms of technical systems and in the embedding human social organizations and institutions.

For example, health care quality and cost-effectiveness can potentially be greatly improved by effective use of information technology on a large scale. Agent-based systems have the potential to offer greater flexibility, enhanced functionalities, and better robustness, reliability, and security, compared to conventional information systems. Patients, family members, and healthcare professionals in hospitals, clinics, pharmacies, and so on, could be supported in their interactions and decisions by various kinds of software agents personalized to meet their information and communication needs. Agent-oriented methodologies can offer the higher level of abstraction needed for this new conception of software.

A critical factor in the successful development of such systems is the understanding of stakeholder needs and wants, how the technologies might alter their relationships, the facilitation of their negotiations, and the communication of those needs to system developers. We need to raise new questions that are not considered in conventional methodologies to assess systems development methodologies for these more challenging types of environments, for example: how well does the methodology support reasoning about autonomy and pro-activeness during the early stages of requirements elicitation?

At first glance one might postulate that object orientation, in its *de facto* standard form – UML/RUP [5], [6], cannot support the above mentioned needs. However, one could also argue that those needs can be addressed to some extent with UML/RUP, even if only partially. Conversely, it is equally necessary to clarify to what extent an agent-oriented method such as Tropos can indeed support understanding and modeling complex domains.

One way to shed light on the above questions is to apply the different methodologies to a common example, which serves to provide a stable and coherent base for discussion and exchange of ideas and results. This type of example is commonly referred to as an “exemplar”.

Yu and Cysneiros have recently proposed the use of an “exemplar”[1] for evaluating methodologies. The exemplar is intended to be used by methodology developers to explore where the strengths and weaknesses of their methodologies lie. The exemplar also aims to help potential users to evaluate different methodologies in depth, and thus to determine how well each would suit their particular needs. It may also help methodology developers to better contextualize their work with respect to other approaches.

Unlike some other exemplars such as [7],[8],[9], this exemplar is intended to be rich and complex enough to test the methodology to its limits. It focuses on a single problem from the health care domain embodying real-world issues and challenges. It was designed to be neutral with regard to any methodology one might be testing. The exemplar is presented in detail in [10]. By having a

rich and complex example, we expect to be able to deeply evaluate each methodology on complex properties that could be otherwise unfairly judged. For example:
(i) How is agent autonomy supported by each methodology? – By exploring the complex problem setting, we discover the challenges faced by each methodology in dealing with autonomy in human and software agents.
(ii) How are non-functional requirements (or quality attributes) addressed in each methodology? – The exemplar presents real-world challenges such as privacy, security and safety.
(iii) How is sociality supported by each methodology? – A rich and complex exemplar can better expose problems regarding sociality that would have been missed if simple examples with few participants were used.
In Section 2, we detail the research methodology we used for using an exemplar approach to compare methodologies. Section 3 presents some of the key findings this study discovered using a sampling of answers to the evaluation questions provided by the exemplar. Section 4 concludes the work with a discussion.

2. Methodology

We applied the exemplar technique proposed in [1] to UML/RUP[5], [6] representing the object-oriented approach and to Tropos[4] representing the agent-oriented approach. Although the specific exemplar provides scenarios and evaluation questions to cover all software development phases, in this work we limit ourselves to those concerning requirements engineering. Tropos was chosen as the agent-oriented methodology because of its familiarity to some of the authors. Previous work applying the exemplar to other agent-oriented methodologies [11] suggests that similar findings could result if we choose another methodology.
The exemplar is based on the Guardian Angel Project[12]. A set of "guardian angel" software provides automated support to assess patients with chronic diseases such as diabetes or hypertension, integrating all health-related concerns, including medically relevant legal and financial information, about an individual. The exemplar builds on software agents representing the hospital (GA_Hospital), the family members at home (GA_Home) and the patient being monitored (GA_PDA). This personal system helps track, manage, and interpret the subject's health history, and offers advice to both patient and provider. The system maintains comprehensive, cumulative, correct, and coherent medical records, accessible in a timely manner as the subject moves through life, work assignments, and health care providers.
The Guardian Angel Project was chosen as the basis for the exemplar as it presents a complex problem that encompasses many of the needs encountered in systems today. It forces methodologies to confront problems such as

distribution, privacy, autonomy, pro-activiness and sociality. As a comprehensive and open-ended problem setting, it enhances the chances for the exemplar to expose strengths and weaknesses of the methodologies.

The exemplar [1] is expressed in terms of a set of numbered scenarios (EA0.0 until EA9.0) such as the one below:

> EA4.0- *Abby is uncertain what insulin dose to give this morning as she has a double session dance class at 10:00 and she remembers all too well that she has had mild hypoglycemic symptoms towards the end of even single session dance classes. She draws an exercise symbol spanning 10 to 11:30 on her daily schedule on the GA_PDA interface and then selects the Advise Dose icon. The GA_PDA informs her that she can either keep the dose unchanged if she thinks she can manage a double carbohydrate snack before the dance class or she can reduce her morning dose of insulin by two units of short acting (regular) insulin.*

Together with this set of scenarios, the exemplar also provides a set of evaluation questions aimed to help evaluate how well the methodology supported the modeling of the set of scenarios. The first column of table 1 shows the concepts addressed by these questions. Note that the scenarios are not complete. Some important scenarios were intentionally omitted in an attempt to evaluate how much the methodology drives and supports the requirements engineer to elicit requirements.

We use three different teams to evaluate both UML/RUP and Tropos approaches. The three teams worked separately. Team one and two were composed of fourth year undergraduate students and one researcher. The researcher is experienced both in using and teaching UML/RUP. All the

	Tropos	UML/RUP
QA1 - Proactiveness	S	W
QA2- Human Autonomy vs software autonomy	S	W
QA3 - Autonomy reasoning	S	W
QA4 - Different levels of Abstraction	S	S
QA5 - Identifying participants in the domain	S	S
QA6 - Capturing, understanding and registering terminology	W	W
QA7 - Domain analysis	S	N
QA8 - Finding requirements	S	N
QA9 -Human-machine cooperation	S	N
QA13 - Reasoning about different non-functional aspects	S	N
QA15 - User interface design	S	S
QA19 - Eliciting and reasoning about Non-Functional aspects	S	W
QA28 - Formal Verification and Validation	W	W
QA29 - Project Management	N	S
QA30 - Working in distributed teams	S	S
QA31 - Tool support	N	S
QA32 - Learning curve	W	S
QA33 - Integration with other methodologies	N	S
Total S (Strengh)	12	8
Total N (Neutral)	3	4
Total W (Weakness)	3	6

Table 1 – Evaluation Questions Graded

undergraduate students have participated in real life software development using UML. Team three was composed of one Ph.D. student with experience in both using and teaching UML/RUP and a fourth year undergraduate student. The exemplar was applied to one methodology at a time. Teams one and three applied the exemplar first to UML/RUP and then to Tropos. Team two applied the exemplar first to Tropos and only then to UML/RUP. Before applying the exemplar each team spent some time studying the approach and applying it to small examples (only for Tropos). All the teams started from scratch for both approaches stopping just before implementation. For each approach they defined all applicable models. Each team then answered the evaluation questions provided in the exemplar. Each question was then rated as "S", "N" or "W". "S" means that the question is strongly supported by the approach, i.e., the methodology provide enough constructs, mechanisms and guidance. "N" means that, although the approach supports the aspect addressed by this question, it does so only up to some extent and therefore, only partially provides ways to model this aspect. "W" means that the approach either does not provide any constructs, mechanisms and guidance or provide very few, hence the requirements engineer would hardly be able to deal with the aspect raised by this question. Figure 1 presents an SADT illustrating the process.

The answers were categorized to help identify the strengths and weaknesses of each methodology and also possible differences among the teams and then to consolidate the answers.

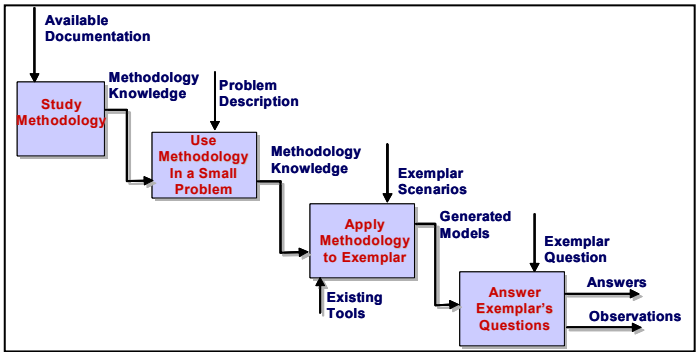

Figure 1 – SADT for the Research Methodology

After all the teams finished answering the evaluation questions and filling in the table, we compiled all the answers to establish an agreement among the groups. Naturally, some of the answers received different qualifications from different teams. For these situations, we analyzed each written comment and the respective models where applicable, and in some cases went back to the teams. We then asked each team if they could further elaborate on the answers in order to clarify doubts arising from comparing their answers with the other teams without disclosing to them the nature of the discrepancy. At the end, we

could agree to one final evaluation (S, N or W). The findings are summarized in Section 3. The complete set of answers and models will be posted at [10]. In carrying out this last step of compiling the answers, we tried to compensate for possible discrepancies due to the use of different teams with possibly different levels of skills and expertise. However, we were surprised that there were not as many discrepancies as we had at first anticipated.

3. Key Findings

In this section, we present some of the key findings illustrating them with answers to corresponding questions. Table 1 summarizes the results based on the teams' conclusions about how strongly or weakly the two approaches responded to the questions to assess each scenario. As mentioned before, while compiling the answers, sometimes the teams were pushed to evaluate their models against those produced by another team, and to reassess their evaluation if necessary.

Autonomy:

UML modeling does not favor dealing with human and software autonomy as can be seen from question QA2. We can see from the answers showed below that although UML offers a few constructs that might initially support some reasoning on human and software autonomy, when the requirements engineer needs to fully understand the implications of allowing humans and software to be autonomous, UML/RUP does not support it. On the other hand, Tropos has several constructs that allow the requirements engineer to reason about autonomy.

- *QA2 "Human Autonomy vs. software autonomy" - In scenario 4.1 Abby (a human being) has the autonomy to follow or ignore advices from the GA and to modify the GA-PDA authorization to communicate with her parent's desktop computer. How would the software engineer handle this autonomy using this methodology? How does one decide which decisions are to be made at design-time and which at run-time?*

Tropos: The requirements engineer is supported to assess this scenario by using the Actor diagram. For example, showing the dependencies between the patient and the GA-PDA, the Actor Diagram allows the requirements engineer to elaborate on social relationships. By representing critical dependencies we were able to visualize that failing to provide autonomy to the patient when the GA_PDA is to report problems would jeopardize the use of the GA_PDA by young patients. This helped us in elaborating solutions to this problem.

The requirements engineer can for example choose which decisions will be made at run-time. This can be detailed by using goals, softgoals and plans in the Goal Diagram. In architectural design, the agents and sub-agents are defined and for each actor and agent we have to identify the capabilities. In detailed design, the plan diagram can be defined in the capability diagram and plan diagrams. Graded Strength.

UML/RUP: Autonomy implies independent action as well as behavior that is not fully predictable. The UML activity diagram is similar to a flow diagram and each agent could be placed into a different swim lane. The predictable interactions among humans and software components could be represented as tasks and decisions. The UML sequence diagram could also be used to specify the message flows between agents. However, in both types of diagrams, the requirements engineer would have to generate an extensive and comprehensive list of all the possible behaviors. Therefore, UML/RUP does not allow the requirements engineer to represent incomplete or partial knowledge about system behavior. Here for example, it was not possible to identify ways to guarantee the needed autonomy for young patients not to have the GA_PDA reporting every problem in the treatment. Graded Weakness.

Learning curve:

Learning curve was another aspect that produced some interesting results. At first, it was thought that UML/RUP could not be graded as strength in learning curve due to its complexity and the numerous models and artifacts. However, we can see from the answers below that although this may be true, the fact that UML/RUP has become a *de facto* standard, and in consequence is being taught intensively, it was easy for the teams to further explore UML constructs and to apply them to the exemplar. It is true that one can argue that to be completely fair we should have used teams with no experience in both methodologies. However, this is almost impossible today, at least if we want to use people with a minimum level of experience modeling systems. It was also revealing that the teams considered Tropos to be weak regarding the learning curve. Being an approach with only a few constructs, one would expect it to be easy to learn and use. However, the answers suggest that due to complex syntax and semantics, together with inadequate documentation (which is not unusual as Tropos is not a commercial methodology), the learning curve can be disappointing. However, it is reasonable to expect that this will change in the future at least regarding the documentation. Although we agree in principle that Tropos can be complex regarding syntax and semantics involved in its constructs, it is not clear at this point to what extent this is due to the lack of good documentation. The responses for the two approaches are summarized as follows.

- QA32 - **Learning curve-** How easily can the requirements engineer learn the methodology and its tools?

TROPOS: Tropos is a complex and robust methodology. Requirements engineers who are new to Tropos should exercise caution when going from one phase to another. Although it has few constructs, they are not easy to grasp both in terms of syntax and semantics. Documentation can be hard to follow sometimes. Graded Weakness.

UML/RUP: It is hard to make a point about this since UML/RUP was already familiar to participants before this work. UML is one of the most well known modeling languages. It is easy to find personnel with expertise on UML/RUP. Thus, in general the learning curve would not be an issue. Graded Strength.

Non-Functional Requirements:

As one might anticipate, UML modeling was evaluated as being weak on questions related to non-functional requirements (NFR). Although we can find work proposing alternatives to handle NFRs together with UML models [16], they have not been incorporated into UML proper and were therefore not considered. In today's practice, NFRs may be manifested in UML models only after they have been operationalized, i.e., converted from design goals into class definitions and operations. UML/RUP does not support the process of reasoning about design goals and how to arrive at operationalizations. In contrast, Tropos, which incorporates the treatment of NFRs as one of its fundamental principles, was evaluated as being very supportive of NFR. The softgoal construct and the different ways to reason and evaluate different alternatives to satisfice one or more NFRs allow the requirements engineer to deeply evaluate alternatives to satisfice NFRs.

- QA19 **Eliciting and reasoning about non-functional aspects-** Possible catastrophes are mentioned in scenario EA6.1, how does the methodology facilitate the elicitation of such requirements?

Tropos: Tropos uses softgoals to model desired properties of the system from the earliest stages of requirements analysis. Security and privacy were among the most important requirements of the system, leading later to the specification of the Security Broker. Permissions and information synchronization were defined to help the system to recover. Graded Strength.

UML/RUP: UML/RUP does not offer a way of representing trade-offs among non-functional requirements. In health care systems, security and privacy are both critical, but may conflict with each other. UML models can

be used to express security and privacy mechanisms once they have been decided upon, but not how the decisions were arrived at. The UML deployment diagram offers only a static view of the system components interface, but this is not enough to describe the complex communication patterns across systems. Therefore, some design decisions remain implicit in UML diagrams. Graded Weakness.

Tool support:

As could also be expected, UML modeling is graded strong on tool support while Tropos is graded Neutral. Having several tools available (even some good freeware such as Visual Paradigm [13]]) tool support was not a problem for UML models. On the other hand, although there are two different tools for Tropos (OME Tool [14] and GR Tool [15]), none of them are mature enough to fully support the requirements engineer. Graded Weakness.

Eliciting Requirements:

One interesting difference among answers from the three groups could be found regarding question QA8 Finding Requirements. All teams rated Tropos as having it as strength, but only one of the groups rated strength for UML/RUP. One possible explanation may lie in the fact that the team rating strength for UML/RUP has first applied the exemplar to Tropos and only later to UML/RUP. It could be argued that applying it first to Tropos led them to deeply explore requirements thus when they applied UML/RUP they were already aware of many needs that could have been missed using UML/RUP. Note that one of the groups applying the exemplar to UML/RUP first, reported they went back to UML/RUP after applying it to Tropos to correct some representations as well as to add others, resulting in a different set of requirements.

In fact, one can argue that the above mentioned situation may come as a weakness of using the same exemplar to evaluate both approaches. By having one team applying two different approaches to the same problem may raise the question that this team would be familiar with the problem when applying it for the second time. Although it is true, it does not diminish the fact that even knowing the problem, the requirements engineer has yet to be able to represent and reason about it using both approaches. That is when the difficulties arise. Not infrequently, the teams faced the problem that they knew what they want to model but were not capable of efficiently doing that using the approach they were testing at the time. On the other hand, applying the same problem to both approaches allowed the teams to better understand strengths and weaknesses of each approach. In fact, this was noticed while we were compiling the answers when a few of them were changed by the teams after we asked them to reassess their answers due to discrepancies among answers from different teams. At this point, they felt the need to compare

experiences from both approaches and sometimes realized that even though they may have graded one approach as strength, in fact, they could do much better with the other approach and therefore the approach initially graded strength should be now graded neutral.
It was particularly interesting that one of the participants with vast practical experience using UML models reported to us after the experiment stating: "Before participating in this work I felt UML models could handle any type of system. Today, I am positive that complex systems such as the one used here can not be efficiently modeled using UML/RUP".
It is also important to mention that all teams reported that Tropos has allowed them to better elicit requirements for the system in study. Since Tropos supports modeling the system-to-be as in UML/RUP but also supports modeling the whole domain and the possible alternatives to stakeholders needs, the teams could better model and reason about possible solutions they could not visualize while using UML models. Moreover, they felt extremely useful to be able to model the "whys" behind the system requirements. They pointed out it really helped them to understand why many of the requirements were needed and to which extent they should push these requirements. Note that none of the teams had previous experience with health care systems. Thus, many of the needs expressed in the exemplar were quite unfamiliar to them; being pushed to understand stakeholders' goals as well as being able to model these goals helped them to contextualize these requirements and hence to better evaluate why these requirements were needed.

- QA8 **Finding requirements -** How does the methodology help in discovering and refining requirements?

Tropos: The early requirements phase of Tropos focuses on finding stakeholders' needs, even before conceiving of the system, leading the requirements engineers to subsequently develop a solution that will respond to the stakeholders' needs.
By using the "Actor Diagrams" and the "Goal Diagrams" the requirements engineering can represent not only the software-to-be but also all the social aspects involved in the problem. For example, we were able to model and reason about the relationship between the parents and the patient while we were also reasoning about how the patient expects to be assessed by the physician. Since Tropos does not restrict us to only represent the software being developed, we could identify and understand many needs we could not using UML modeling. For example, the need for having flexible plans for exercise and diet was essential to be assessed and could only be seen because of the dependencies among actors and the ability to model and reason about non-functional requirements. Graded Strength.

UML/RUP: Supports the identification of user's requirement mainly through use case diagrams. The requirements can be refined in further design steps, so a use case could be split into two or more use cases. The problem is that these requirements are seen as system processes. In other words, they represent actions the system must perform in order to respond to actors' inputs. There are no semantic distinctions between a task, a goal or a soft goal. And also it is difficult to analyze a domain full of "maybe situations", intentions, assumptions and user's expectations. For UML/RUP, expectation is an order or it is not taken into account. Once more, this could be represented by the use of stereotypes in the use case diagram, but it is not a widely used standard.
By the same token, UML modeling did not support us to identify possible requirements that could arise from relationships between actors such as the parents and the physician which may not at first be connect by any software function. However if we can model and reason about this relationship we could have realized the need for many software requirements such as having the possibility for the GA_Home to negotiate with the GA_Physician to re-schedule existing appointments or to schedule a new one only with parents if they feel insecure about the treatment but may not want to alarm their son/daughter. Graded Neutral.

4. Conclusion

This work presents a practical evaluation comparing to what extent the emerging agent/goal-oriented approach and the widely adopted object-oriented approach would support requirements engineering for today's complex application environments. An exemplar from the health care domain was used to exercise both approaches as much as possible. By doing so, we were aiming at evaluating these two approaches in light of the current needs for developing today's complex software. This work is part of a broader project that aims to experiment with and to refine the exemplar proposed by Yu and Cysneiros [1] while learning about strengths and weaknesses of different methodologies. To date, aside from UML/RUP and i*/Tropos we have applied the exemplar to GAIA [18] and Message [19].
The results, summarized in Table 1, indicate that although object orientation can cope with complex domains to some extent, it is nevertheless not sufficiently effective for representing and analyzing the complex situations that we face today. On the other hand, while agent/goal orientation is better able to respond to these challenges, it suffers from a number of pragmatic concerns such as learning curve and availability of efficient tools.
For future work, we plan to investigate how methodologies that evolve from UML such as AUML [17] would evaluate using this exemplar. We also aim to apply this exemplar to other goal-oriented approaches such as GBRAM[2]

and KAOS[3] to confirm our expectations that they would evaluate very close to Tropos.

5. References

[1] E. Yu and L.M. Cysneiros, "Agent-Oriented Methodologies - Towards a Challenge Exemplar," Proc of the 4th Intl. Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS'02), Toronto, May 2002.

[2] A. I. Antón. "Goal-Based Requirements Analysis," Second IEEE International Conference on Requirements Engineering (ICRE '96), Colorado Springs, Colorado, pp. 136-144, 15-18 April 1996.

[3] A. Van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," Proc. of 5th IEEE Int. Symp. on Requirements Engineering, 2001.

[4] P. G. Bresciani, Giunchiglia, F., Mylopoulos, J., and Perini, A., "TROPOS: An Agent-Oriented Software Development Methodology," Journal of Autonomous Agents and Multi-Agent Systems, 8, 203–236, 2004..

[5] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual: Addison-Wesley, 1999.

[6] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1998.

[7] D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", Communications of the ACM, vol. 15, pp. 1053 - 1058, December 1972.

[8] M. Feather, Fickas, S., Finkelstein, A., van Lamsweerde, A., "Requirements and Specification Exemplars," Automated Software Engineering, vol. 4, 1997.

[9] T. Olle, "Information Systems Design Methodologies: a comparative review," Proc. IFIP WG8.1 CRIS 1, North-Holland, 1982.

[10] http://www.cs.toronto.edu/km/aometh/

[11] Cysneiros,L.M., Werneck, V. and Yu,E. “Evaluating Methodologies: A Requirements Engineering Approach Through the Use of an Exemplar “ in Proc. of 7th Workshop on Requirements Engineering, Tandil, Argentina, Dec 2004 pp:40-55.

[12] P. Szolovits, Doyle, J., Long, W.J., "Guardian Angel: Patient-Centered Health Informa-tion Systems," http://www.ga.org/ga/manifesto/GAtr.html".

[13] Visual Paradigm, at "http://www.visual-paradigm.com."

[14] OME3 Tool, at "http://www.cs.toronto.edu/km/GRL/."

[15] GR Tool, at "http://sesa.dit.unitn.it/goaleditor/"

[16] L. M. Cysneiros and Leite, J.C.S.P., "Non-Functional Requirements: From Elicitation to Conceptual Models," IEEE Trans. Soft. Eng., vol. 30, Number 5, pp. 328-350, 2004.

[17] J. Odell, M. Nodine, and R. Levy, "A Metamodel for Agents, Roles, and Groups", 5th Intl. Workshop on Agent-Oriented Software Systems (AOSE), NY, July 2004, LNCS vol. 3382, Springer-Verlag, Berlin, 2005.

[18] Zambonelli, F., Jennings, N. R. and Wooldridge, M., Developing Multiagent Systems: The Gaia Methodology, ACM Transaction on Software Engineering and Methodology, Vol. 12, No. 3, July 2003, pp 317-370.

[19] MESSAGE, http://www.eurescom.de/~public-webspace/P900-series/P907/index.htm, May 23, 2000.