Requirements Engineering for Large-Scale Multi-Agent Systems

Luiz Marcio Cysneiros [1] and Eric Yu [2]
[1] Department of Mathematics and Statistics
York University
cysneiro@mathstat.yorku.ca
[2]Faculty of Information Studies
University of Toronto
yu@fis.utoronto.ca

Abstract. Large-scale software systems typically involve a large number of actors playing different roles, interacting with each other to achieve personal and common goals. As agent-based software technologies advance, systematic methods are needed to support the development of large-scale multi-agent systems. As with other kinds of software systems, successful system development relies on in-depth understanding of stakeholder needs and wants, and their effective translation into system requirements, and eventually into executable software. This paper presents a requirements engineering methodology based on agent concepts at the requirements modeling level. The strategic actor is used as the central organizing construct during requirements elicitation and analysis. In considering alternative arrangements of work processes and system interactions, strategic actors seek to exploit opportunities and avoid vulnerabilities. The methodology starts by building a lexicon as a preliminary step. The relevant actors are then identified. A breadth coverage step produces a first-cut model of the domain and the social relationships within it. The models are then developed depth-wise to capture organizational and individual goals and to explore alternatives. The methodology complements and extends the ***i**** modelling framework. By taking into account agent characteristics such as autonomy, intentionality, and sociality starting from the requirements level, the methodology leads naturally into the development of large-scale systems that employ multi-agent software technologies. An example from the healthcare domain is used to illustrate the methodology.

1. Introduction

Imagine the widespread use of software agents to support healthcare on a large scale. Healthcare professionals, patients, and even family members would be supported in their interactions and decision making by various kinds of software agents personalized to meet their information and communication needs. Information technology is widely recognized as a key ingredient for improving healthcare quality and cost-effectiveness. Compared to conventional information systems, agent-based systems have the potential to offer greater flexibility, enhanced functionalities, and better robustness, reliability, and security.

While many elements of agent technologies are advancing beyond experimental stages, methods for building large-scale real-world applications are only beginning to be developed [19] [20]. Such systems are characterized by large numbers of players, with complex relationships, and often conflicting goals. Humans, hardware, and software interact in much more intricate ways then in conventional systems which

automate routine work processes. A critical factor in the successful development of such systems is therefore the understanding of stakeholder needs and wants, how technologies might alter their relationships, facilitation of their negotiations, and communication of those needs to system developers.

Requirements engineering is an area of growing importance as it focuses on identifying and characterizing "what" the system should do, while subsequent software engineering stages elaborate on the technology-oriented "how". Various approaches, including models, languages, methodologies, and tools have been developed to assist and support requirements activities such as elicitation, analysis, negotiation, verification and validation [10].

This paper presents a methodology that uses the strategic actor concept as the central organizing construct. Unlike traditional requirements modeling techniques (e.g., [5] [13]) that assume well-defined activity steps, input-output flows, or object interactions, strategic actors may act autonomously within the constraints of their social relationships with other actors. As new technologies and work arrangements are being proposed, various players explore alternatives and seek to advance or protect their strategic interests as they face uncertainty and turbulence in their environments.

The methodology builds on the ***i**** framework for agent-oriented requirements engineering [15][16]. This paper extends earlier work by introducing the use of a lexicon control as a front-end to guide model construction, as well as systematic steps on how to use the i* framework to elicit requirements. A broader methodology based on ***i**** covering the full range of software engineering activities is outlined in [2]. This present paper elaborates on the very early stages of software development.

By incorporating agent characteristics such as autonomy, intentionality, and sociality into the earliest stages of requirements analysis, the methodology facilitates the development process for large-scale multi-agent software systems. For detailed analysis, strategic actors are refined into agents that play roles or occupy positions. Each of these may be used to represent composite actors such as organizational units consisting of multiple positions, roles, and agents. Composite actors may have goals that complement or conflict with those of individual actors.

The methodology is illustrated with an example from the healthcare domain. The scenario is the provision of automated support to assess patients with chronic diseases through the use of a set of "guardian angel" software agents integrating all health-related concerns, including medically-relevant legal and financial information, about an individual. This personal system will help track, manage, and interpret the subject's health history, and offer advice to both patient and healthcare provider. The example setting has been proposed as an exemplar for advancing research in agent-oriented software development methodologies [17], and is based on the Guardian Angel project proposal [14]. Although the methodology could be presented more concisely with a simpler pedagogical example, we have chosen to use a realistic example to illustrate the complexity of issues that can arise in a large-scale application.

2. Agent-Oriented Modeling for Requirements Engineering

Most requirements modelling techniques have focused on the specification of processes in terms of activity steps, input and output flows, or interactions and message exchanges among mechanistically-behaving objects (e.g., [5], [13]). Agent-based systems function in environments where humans, as well as hardware and software, have considerable autonomy. Thus their behaviours are not so well defined or predictable. Instead, they are better characterized by intentional concepts – goals, beliefs, abilities, and social relationships such as interdependencies and commitments.

The ***i**** modelling framework [15] was developed to introduce an intentional and social ontology for requirements engineering. Systems and their environments are described in terms of intentional relationships among strategic actors. Actors are taken to be intentional in that they have goals, beliefs, etc., and strategic in that they seek to exploit opportunities and to mitigate vulnerabilities. *Actors* may be abstract (*roles* defining responsibilities), concrete (*agents* – human and non-human individuals or classes with specific capabilities), or other organizational constructs (e.g., *positions* which package a number of *roles* together to be assigned to a single concrete *agent*). Composite actors may be composed of constituent actors.

External relationships among actors are expressed in the *Strategic Dependency* (SD) model. Actors depend on each other for goals to be achieved, tasks to be performed, resources to be furnished, and softgoals to be satisficed[1]. Internal relationships among the intentional elements within an actor's reasoning are expressed in the *Strategic Rationale* (SR) model. Rationales are modelled through means-ends relationships, task decompositions, and softgoal contributions. SD models are used to represent different configurations of relationships, for example, representing different ways in which multi-agent systems can be used within a healthcare context, whereas SR models are used to make the reasoning about such alternate configurations explicit.

The ***i**** concept of strategic actor is an abstraction used to hide the detailed actions within the actor's discretion. Strategically significant elements of work processes are described in terms of dependency relationships among actors. The agent-oriented modelling approach offers a higher-level description that is a more faithful representation of reality as real-world agents frequently depart from standardized routines and procedural process specifications.

Note that during the requirements stages, agent concepts are used to encompass both human and non-human agents. The agent-based ontology is used to bring out complex social relationships, so as to expose strengths and vulnerabilities in the existing process. This allows the stakeholders and analysts to conceive of alternatives to the existing process, guided by goals and evaluation criteria. Such alternatives might, and often does, introduce software agents into the process.

---

[1] We use here the same notion used in [3] that an NFR can rarely be said to be satisfied. Goal satisficing suggest that the solution used is expected to satisfy within acceptable limits. The term satisfice was coined by Hebert Simon to express "good enough" alternatives

3. The Methodology

An initial step aims to develop an informal understanding of the domain by systematically collecting and analyzing the vocabulary used by stakeholders. A Lexicon is constructed using supporting tools. The lexicon is then used to help construct an agent-oriented model of the social structure of the domain. A first-cut Strategic Dependency (SD) model is then constructed to provide an overview of the existing processes and related systems. As the elicitation proceeds, the first-cut model provides a scaffolding for stakeholders to voice their concerns and aspirations, critique the current processes and organizational arrangements, and to articulate possible new arrangements. These elaborations will be used to construct more detailed models where the achievement of *goals* are evaluated, leading stakeholders to explore alternate systems and organizational arrangements which will allow them to achieve their *goals* more effectively. Figure 1 illustrates the process as an SADT (Structured Analysis and Design Technique) model [12]

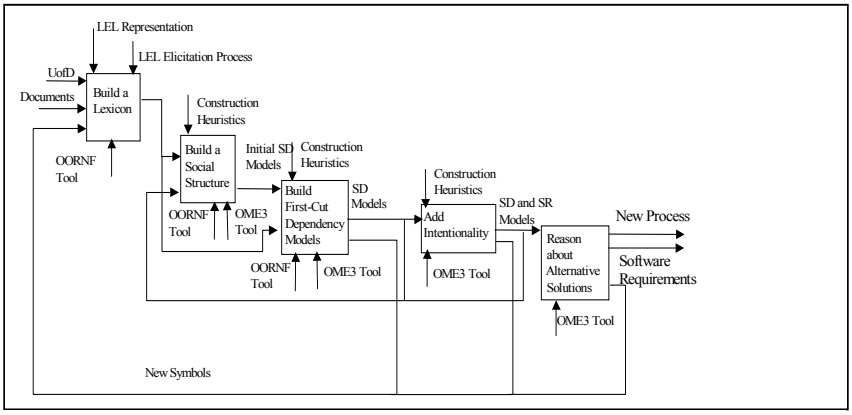

Figure 1 – An SADT Model of the Overall Methodology

The methodological steps presented in this paper address primarily the early stage in requirements elicitation and analysis. During this stage, one develops an understanding of the problem by analyzing organizational structures, processes, and social relationships. Alternatives to the existing process are explored, including the possible introduction of software agents, and how these different alternatives might contribute to overcome problems and improve the process.

A subsequent stage in the methodology, not illustrated in this paper, addresses the late requirements where the system-to-be is introduced and therefore its requirements are to be detailed, modeled and analyzed in the presence of quality requirements (also known as Non-Functional Requirements [1]). Tradeoffs will have to be made, some of them requiring one more time social and organizational concerns to be brought to light.

3.1 Understanding the Vocabulary

Dealing with complex domains demands to first understand the vocabulary used in the domain. Clarifying the vocabulary can help avoid many problems typically caused by the naive assumption that it is easy to understand what the stakeholder is talking about [4][6]. The methodology starts by constructing a controlled vocabulary to act as front-end support for the ***i**** modelling process, as an ongoing resource for the project, as well as for future reuse. It serves to facilitate and focus communication with stakeholders and among development team members. LEL, Language Extended Lexicon proposed by Leite [9], is used to capture the vocabulary used in practice in the domain. Its objective is to register the vocabulary of a given UofD[2]. It is based upon the following simple idea: understand the problem's language without worrying about deeply understanding the problem. LEL is mainly used to register terms (words or phrases) relevant to a specific field of application.

LEL is based on a controlled vocabulary system composed of symbols where each symbol is an entry expressed in terms of notions and behavioural responses. The notions record the meaning of the symbol and its fundamental relations with other entries. The behavioural responses specify the connotation of the symbol in the UofD. Each symbol may also be represented by one or more aliases and will be classified as a subject, a verb or an object.

The construction of the lexicon is guided by the principles of minimal vocabulary and circularity. The circularity principle prescribes the maximization of the usage of lexicon symbols when describing lexicon entries, while the minimal vocabulary principle prescribes the minimization of the usage of symbols exterior to the lexicon when describing lexicon entries.

First Contact: Open-Ended Interview - To build an initial version of the LEL we start by conducting an open-ended interview [7] with some or all of the stakeholders (depending on their availability, and taking into account possible political sensitivities). During this meeting we will try to get a first idea of the domain and to capture some initial lexicon symbols. We will also try to identify possible documents that we can use to further elicit other symbols. Although we are not yet worried about capturing intentions, if there are already some preconceived notion of a target system, one question must be asked now: Why do you need this system? The answer will be used to delimit the context in which we will focus our work. Try also to find out the main stakeholders involved in the domain. All actors discovered should be included as symbols of the lexicon. The relatively neutral activity of eliciting a vocabulary allows the RE to establish rapport with and among stakeholders, cultivating a collaborative atmosphere for potentially more contentious issues later. This activity also affirms the primacy of the stakeholders in determining eventual outcomes, with the RE in a supporting, facilitating role.

LEL is useful not only for understanding the domain but also as a starting point for creating different models throughout the process. In addition, capturing the vocabulary in an organized form benefits the reuse of the domain knowledge.

---

[2] *"Universe of Discourse is the general context where the software should be developed and operated. The UofD includes all the sources of information and all known people related to the software. These people are also known as the actors in this UofD."*

Domain Knowledge Construction – Start to register all the relevant words or phrases peculiar to the domain using the lexicon. This process leads to a spiral growth of our knowledge about the domain and continues until no new symbols are found.

Validating the Lexicon with Stakeholders – Because the lexicon uses a natural language structure, it can be validated by stakeholders without much difficulty. At every stage, validating the Lexicon with the stakeholders is very helpful in gaining additional knowledge about the domain

3.2 Outlining the Social Structure

To appreciate the complexity in a large-scale application, it is helpful to begin by identifying the main actors and roles and outlining how they relate. That gives the RE a better context for understanding the processes and the political landscapes. To build an outline of the social structure, we will use the lexicon and semi-structured interviews. The social structure is depicted in a model consisting of *actors* and the structural relationships among them (an agent *play*ing some role, *occupy*ing some position; a position *cover*ing some roles; an actor being *part-of* another actor; an actor *is-a* specialization of another actor). *Actors* may be differentiated into *agents*, *roles*, and *positions* but may also be generic, if the distinctions are not clear at this point. For example, Figure 2 depicts the idea that an agent physician person occupies the physician position which in turn may cover either the role of a physician in a hospital or the role of a physician in a practice. Notice that these two roles will certainly encompass different goals, meaning that each one might have different requirements for the software to be built.

Identifying the Main Actors - We start by picking up all the symbols classified as subjects in the lexicon. They are strong candidates to be actors. For each symbol represented as an actor in the model, check if other symbols categorized as subjects

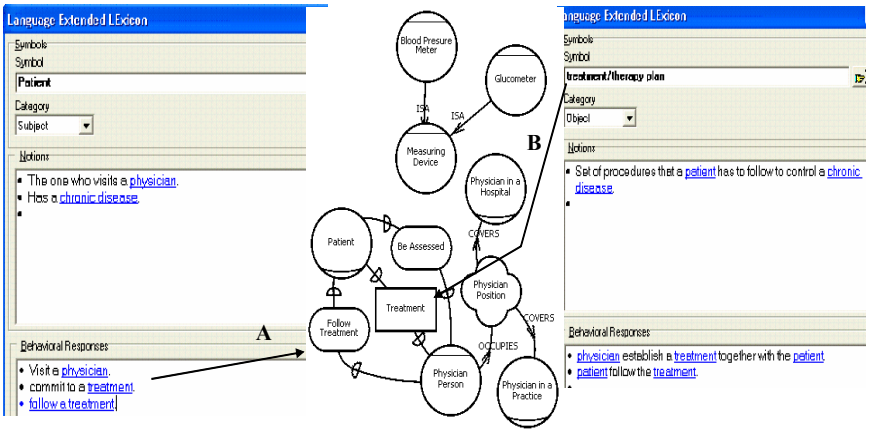

Figure 2 – Finding Goals and Resources Dependencies

appear in the Notions of that symbol in the lexicon. The presence of other subject symbols may suggest the possibility of a structural relationship between these two

actors. Once we do it to all the symbols in the lexicon we should get an initial idea of the domain's social structural relationships.

Validating the Social Structure with Stakeholders - This can be done by using semi-structured interviews where the model will be explained and presented to the stakeholders so they can confirm it or help correct it.

3.3 Building a First-Cut Dependency Model

At this stage we are trying to understand and model the "as-is" processes and systems. Initially, we will be mostly aiming to construct a Strategic Dependency (SD) model [15]. The SD model depicts a process through the use of a network of dependency relationships among *actors*. In ***i****, a *dependency* is a relationship in which one *actor* (the *depender*) depends on another *actor* (the *dependee*) for something (the ***dependum***) to be achieved. A *dependum* can be a *goal, task, resource, or softgoal*, reflecting the types of freedom allowed by the relationship, as detailed below.

Identifying the Main Dependencies - To start building an SD model, we revisit the symbols in the lexicon. Other elicitation techniques (open-ended interviews, document reading, observation and protocol analysis [7]) can also be used to enhance the initial model derived from the lexicon.

For each symbol that has been identified as an actor in the social structure model, check if the behavioural responses have any other symbol that is a subject (another probable *actor*), or have sentences with two symbols that are subjects. These suggest the presence of a *dependency*. Since not all dependencies can be discovered through the lexicon, a complementary approach is to using the social structure model (Section 3.2) to ask each actor in the model: 1) On Whom do you depend to do your job? 2) What other people/devices do you interact with? 3) Who depend on you to do their job? In addition, dependency types may not be directly apparent from the lexicon. Hence, the RE may need to elicit from actors the nature of the dependency relationships. The lexicon can provide further help in the following ways:

Searching for Goal or Task Dependencies - Symbols that are verbs and are present in the behavioural responses of other subject symbols suggest a task or goal dependency. Arrow A in Figure 2 illustrates this heuristic Also, checking every symbol classified as a verb may lead us to many task or goal dependencies, although sometimes we may be seeing a task or goal that is internal to an actor. In that case, we may put this task or goal aside for the moment until we start building the SR models.

In ***i****, a *goal dependency* is one in which one *actor* depends on another to bring about a certain condition or state in the world, while the depended *actor* (the dependee) is free to, and is expected to, make whatever decisions are necessary to achieve the *goal*. Thus it also indicates that one *actor* does not care how the other *actor* will achieve this *goa*l. For example, the Patient depends on the Physician to accomplish the goal of *Being Assessed*. It is a *goal* dependency because the patient does not care specifically how the physician will achieve that *goal*. On the other hand, if the *depender* does care, this *goal* will be represented as a *task*. A *task* dependency means that the *actor* who is delegating this *task* specifies how the *task* is to be performed.

Searching for Resource Dependencies - A *resource* dependency means that one *actor* depends on the other for the availability of an entity (physical or information).

Symbols that are objects and that have one or more other symbols that are subjects in its notions may represent a *resource* dependency between the symbols that are subjects. Arrow B in Figure 2 shows an example of this heuristic.

Figure 3 depicts the First-Cut model.

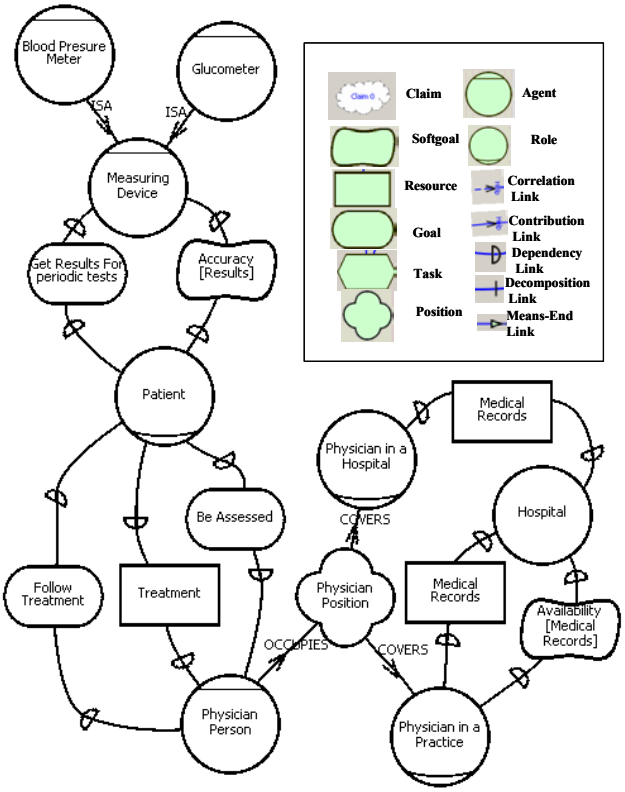

Figure 3 – First-Cut Model

The first-cut dependency model aims to provide a broad-brushed picture of the major relationships and associated processes. *Goals* are recorded as they are recognized, but no attempt is made to further pursue their "hows" and "whys". Understanding the "whys" requires that one have at least a basic understanding of the "whats" (actual process). The same applies for *softgoals*. A *Softgoal* is similar to a goal in that it is a condition in the world that an actor would like to achieve. However, the criteria for the condition being achieved are not sharply defined. Therefore, *softgoals* are satisficed (sufficiently achieved) rather than satisfied [3]. In Figure 5 we can see the *softgoal* Availability regarding Medical Records. This *softgoal* appears in the first-cut model because the Physician Working in a Practice may have major problems for obtaining Patient's Medical Records from Hospitals where the Patient had been visiting before consulting with this Physician. Hence, being a major concern for a Physician Working in a Practice, this *softgoal* can be elicited even if we are not concerned about *softgoals*.

3.4 Elaborating on processes and rationales

At this stage we would have a general idea of the main *actors* and their external *dependencies*. We now need to understand the *actors*' internal implementation of the

processes through eliciting the *tasks* within each *actor* and modelling them in SR models. The Strategic Rationale (SR) model provides an intentional description of processes in terms of process elements and the rationale behind them. SR models describe the intentional relationships that are "internal" to *actors*. The primary focus at this stage is on eliciting *task* elements, without going into the rationales. The elicitation of the *task* elements helps clarify the boundaries of responsibilities among *actors*.

Building the SR model enables us to go into details that not only help understand certain needs but also disclose new *dependencies* among *agents*. To do that, we focus on finding the main *tasks* that each *actor* is responsible for. *Tasks* tend to be easier to identify than *goals*. They are more tangible since the stakeholders or the end-users carry them out as activities. *Goals* and rationales that underlie the business will be the focus of our next step in the methodology. However, if some *goals* are evident at this stage, we record them in the model but will not elaborate on them until later.

Examining the Lexicon - We will, once again, use lexicon symbols classified as verbs to be candidates for *tasks* in SR models.

Behavioural responses in symbols that represent actors frequently hide a *goal* or *task*, while behavioural responses that have an object symbol may disclose a resource dependency.

Drawing Some Scenarios – at this point, scenarios [18] can be helpful in refining *tasks* and expanding on an *actor* (an actor may be understood as any possible instance of an actor, i.e. *agent, role and position*). To expand on an *actor*, we may ask the stakeholders in what possible scenarios could this agent be involved.

The scenario is an informal way of eliciting requirements and is non-intentional. It is more useful in late requirements engineering stage and is playing more of a supporting role at this early stage of requirements elicitation. Be careful to only describe scenarios that are pertinent to the problem and therefore not to expand the UofD more than it needs to be. If one has created the scenario by reading documents, validate each scenario with the stakeholders.

Asking Questions - Also, for each *role* or *position* in the models we may frequently ask: Who are the actors involved with this *role/position*. Furthermore, for each or *actor* we must ask two questions: 1) What is this person responsible for? 2) What are the processes in which this person is involved? Answers to these questions may be directly modeled in a SR model or be the trigger for drawing a scenario.

This process continues until no more *tasks* are found. We confirm with the stakeholders that the process is now correctly understood and modeled. Any new symbol found during the process has to be added to the lexicon and the process repeats looking for new *actors*, new *dependencies* and new *tasks*.

Managing Viewpoints - Stakeholders typically have different viewpoints about one subject matter. Different stakeholders may view the same process in different ways or have different ways of achieving the same *goal*. Different viewpoints can be represented as different models and later consolidated, or expressed as alternative ways of achieving a *goal/softgoal*.

Finally, it is also very important to maintain consistency between SD and SR models as new findings in SR models may arise in this step of the methodology.

Figure 4 shows some of the *tasks* that were found through the use of the above heuristics. We can see for example that in order to Follow a Treatment, a Patient have to

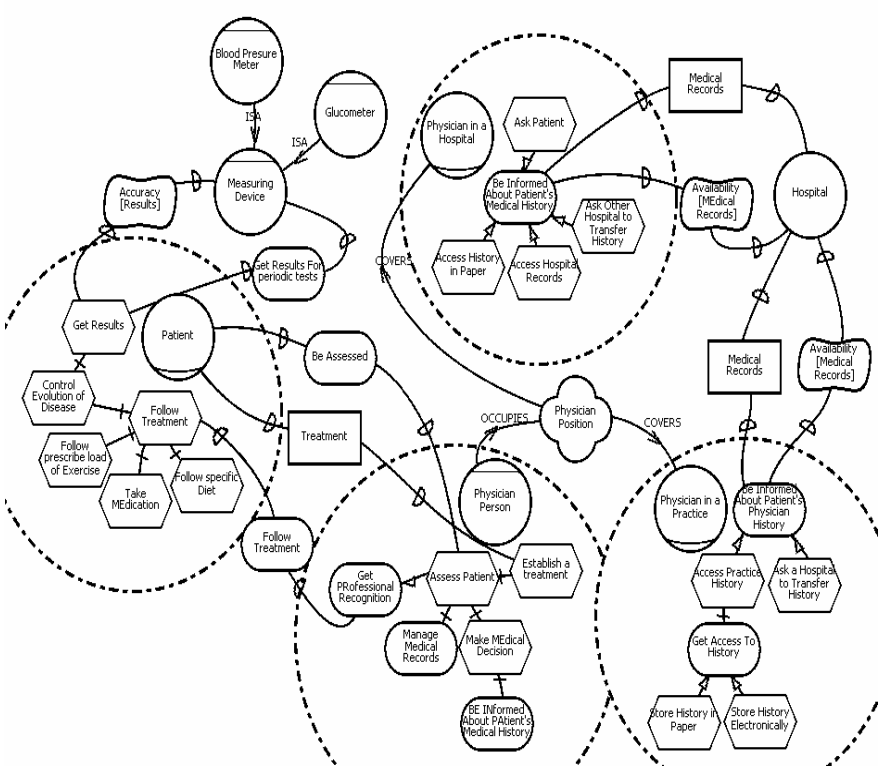

Figure 4 – Elaborating on Processes and Rationales

Control the Evolution of the Disease, Follow a Prescribed Load of Exercises, Take Medication and Follow a Specific Diet. On the other hand we can see that a Physician to Assess a Patient has to Manage Medical Records, Make Medical Decisions and Establish a Treatment. The *goal* Get Professional Recognition appears in this model because we arrived to this *goal* because it was not clear why a Physician depends on a Patient to Follow the Treatment. As happened with the Availability *softgoal* in the First-Cut model, although at this stage we are not yet concerned about intentions, since we got to know a *goal* we have to model it. Later we can further reason about this *goal*, for now it is enough to represent it in the model.

3.5 Elaborating on the Intentional Dimension

Once we have a model representing the process as-it-is today, we elicit the *goals* that underlie the process, why the process is the way it is, what are the *goals* that have to be achieved so the business can be successful. Here we will be seeking higher-level *goals* that will represent among other things the business objectives. We will also be looking for individual *goals*, i.e., *goals* that are particular to an *actor*. In this step we will mostly use semi-structured and open-ended interviews and eventually document

analysis [7]. When using these techniques, one may keep asking the following questions: Why does this *actor* need to perform this *task*? Why does it have to be performed this way? To discover *goals* that are currently not satisfied, one could ask: What is wrong with the process? What are the things you (the *actor*) dislike most? What should be changed? What are the major weaknesses? Aside that, discovering the major strengths in the current process/system would suggest *goals* that are currently satisfied.

It is important to have in mind that this is not a top-down process and neither a bottom-up one. Since we depart from already elicited *tasks*, it is more likely that we start using a bottom-up approach rather than a top-down approach. However, we recognize that in some situations the opposite approach might be more suitable.

At this point in the methodology, it is important to be careful about privacy and political aspects. A stakeholder may be reluctant to disclose negative aspects of a process because he or she may have fears about sanctions or reprisals. Thus, interviews with more than one stakeholder at a time should be carefully handled and better avoided. Also, if one stakeholder says there is no problem with the actual process, while others say the opposite, one could suspect that important personal interests might be at stake. These should be probed further as they may suggest or preclude possible future alternatives.

Aside from *goals*, we will also be looking for *softgoals*. Some *softgoals* can be identified and modeled during the previous phases, but we defer the comprehensive elicitation and elaboration of softgoals until this stage since *softgoals* are typically used as selection criteria in evaluating and suggesting process alternatives. The notion of the *softgoal* elaborated here is related to that of non-functional requirements [3], but our *softgoals* relate to *actors* that can be a human agent or a computer agent.

Following a taxonomy of common *softgoals* during elicitation from the stakeholders can be very helpful. For example, for each *task* and *goal* we may ask: Does this *task/goal* need Safety? Accuracy? Performance? Whenever the answer is yes, we may represent it as a *softgoal* and refine it until we have the operationalizations that will satisfice this *softgoal*. Refinement can be either top-down or bottom-up, and more likely it is a mix of both. Refinements may also be part of the taxonomy and must be used as guidance instead of a mandatory approach.

After we refine all the *softgoals*, we may check for possible interdependencies, (positive and negative contributions, among *softgoals*). To do that, we use the approach used in [4]. This approach proposes three heuristics to search for interdependencies: 1) Evaluate different graphs for the same type of softgoal, e.g., all the graphs related to performance. 2) Evaluate graphs for softgoals that are frequently conflicting, e.g. Usability and Security. 3) If the number of graphs is not too large, pair wising different graphs (of course excluding graphs that have already been compared within the two previous heuristics). Where negative contributions are found, tradeoffs will have to be made among possible alternative solutions.

To check for model completeness at this stage, we check if all *goals* are evaluated. *Goals* are related to tasks through *means-ends links*. The *tasks* are the different ways in which the *goal* can be accomplished. Each *task* may consist of *subgoals, subtasks, resources, and softgoals* (via the *task decomposition link*). All elements of a *task* must be satisfied in order for a *task* to be satisfied. A *goal* is satisfied if any of its

alternative tasks is satisfied (via the means-ends links). Note that (hard) goals can be satisfied, while softgoals can only be *satisficed*.

Softgoals have also to be evaluated through a qualitative reasoning that can be carried out using contribution links among *softgoals*. The semantics of the links are based on the concept of satisficing [3]. The most common contribution types are *Help/Hurt* (positive/negative but not sufficient to meet the parental goal), *Some+/Some-* (positive/negative of unknown degree), whereas *Make/Break* indicates positive/negative of sufficient degree, i.e., strong enough to say that the softgoal is met or not met. Although these distinctions are coarse grained, they are enough to decide whether we need further refinement and search for more specific *softgoals* and

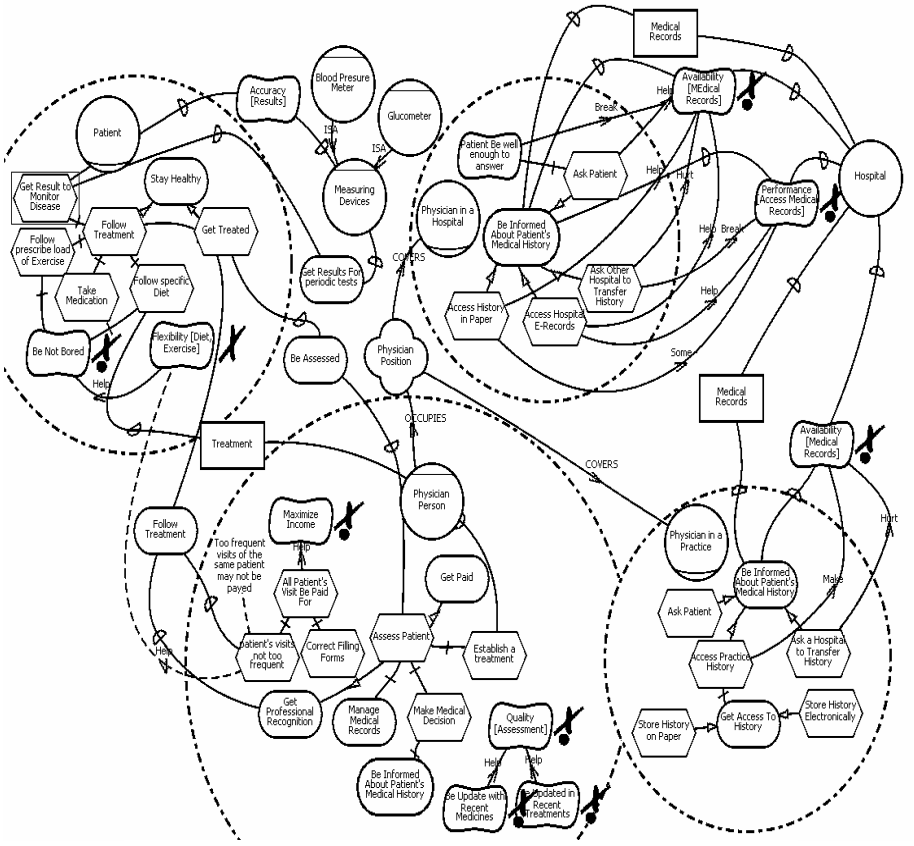

Figure 5 – Addressing the Intentional Level

operationalizations or not. *Contribution links* allow one to decompose *softgoals* to the point that one can say that the operationalizations to this *softgoal* have been reached (i.e., the goals are no longer "soft").

Figure 5 shows part of the model after we addressed the intentional level. We can see for example that a Patient has a *goal* of Staying Healthy, which can be achieved by Getting Treated and Following a Treatment. We can also see that further refinements

addressing the intentional level lead to find out that in order to Follow a Prescribed Load of Exercise and to Follow a Specific Diet, a Patient does not want to Be Bored. This is because an inflexible load of exercise can pushes the Patient to the level of not doing any exercise at all (expressed by the *softgoal* Flexibility with a *help contribution link* to the *softgoal* Be not Bored). The same way, a fixed diet that does not allow one to make changes can lead to frequent occurrences of improper meals, which in turn might lead to the need of changing medications. This may, in turn, lead to frequently visits to the physician. Since Physicians depends on the Patient to Follow a Treatment in order to Maximize Income (Frequent visits may not be paid), having the *softgoal* Be not Bored evaluated as weakly denied clear indicates a point for further reasoning to take place in the next step of the methodology.

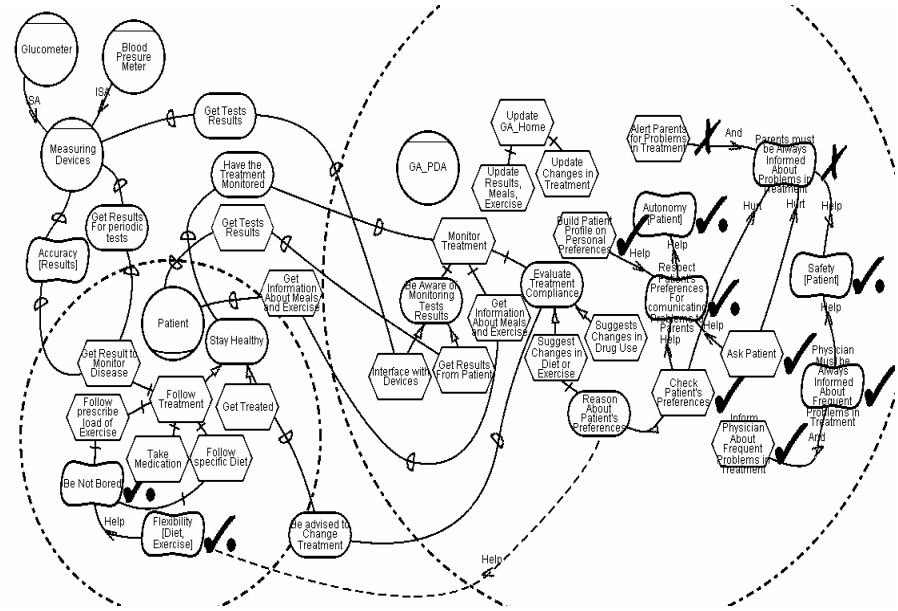

Figure 6 – Introducing the GA_PDA Software Agent as an Alternative

We can also see in Figure 5 that in order to satisfice the *softgoal* Quality regarding the Assessment, the Physician may be Updated with Recent Medicines as well as to be Updated in Recent Treatments (both *softgoals* have a *help contribution link* to Quality). Once these two *softgoals* were considered partly denied because treatments and drugs evolve too fast, the Quality *softgoal* is automatically evaluated as partially denied since none of the *softgoals* that might contribute to its satisficing are satisfied or at least weakly satisfied.

3.6 Exploring Further Alternatives

By this time in the process, a space of alternatives would likely have already emerged for achieving some of the *goals*. We may pay special attention to the problems raised and therefore to *goals* (hard or soft) that are not currently being

achieved. We may look for possible ways of achieving these goals. For example, diminishing the dependency one *agent* has on another may diminish the vulnerability of this *agent* against the other and lead to a more satisfactory process. This is particularly true when one agent is frequently not achieving a *goal* that is delegated by another *agent*. It is also worth considering whether adding new *actors* (either human or software) to the process may help. Software agents can absorb some responsibilities previously performed by a human agent, or can introduce new capabilities.

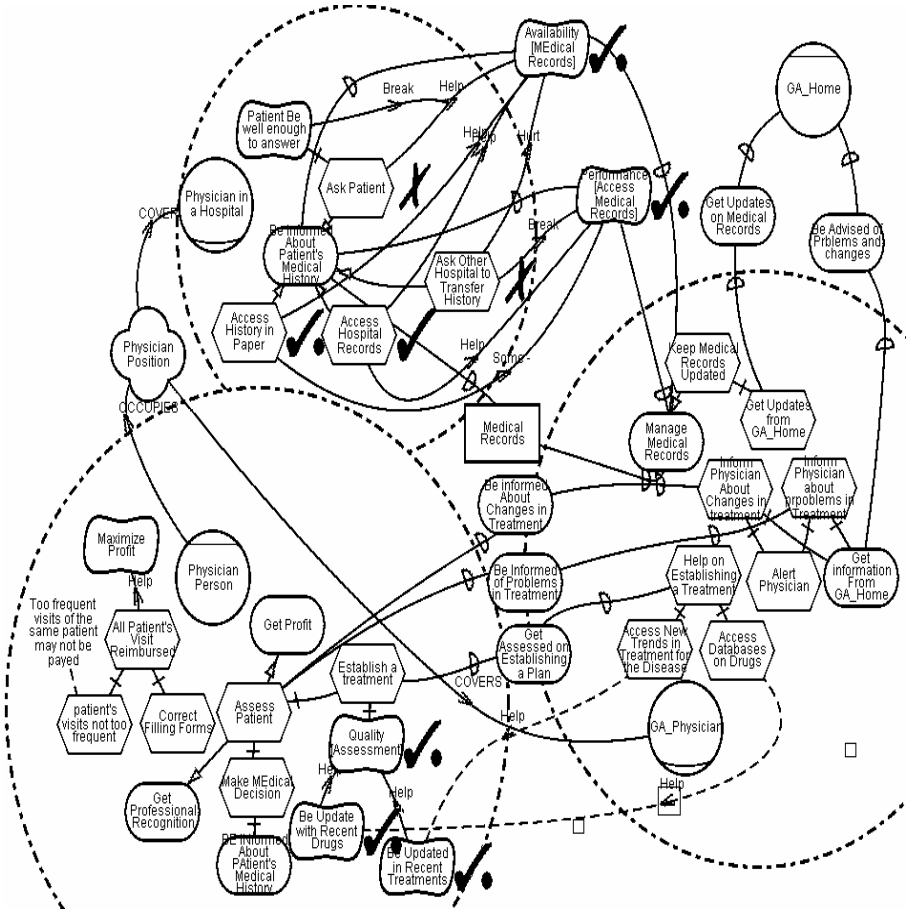

Figure 7 – Introducing the GA_Physician Software Agent as an Alternative

It is also important to compare different viewpoints. Focus on how each viewpoint contributes positively and negatively to achieve the *goals*. For example a Physician in a Practice has a different viewpoint for the *goal* Be informed about Patient's Medical History than the Physician in a Hospital does. Eventually, a compilation of two or more different viewpoints can bring the best solution. If possible, conduct a group meeting with the

involved stakeholders to elaborate on new alternatives. One has yet to be aware about political and personal constraints since different alternatives may lead to different redistribution of power as typically indicated in the stakeholders' personal *softgoals*.

Figure 6 shows the introduction of a new software agent GA_PDA as one possible alternative to the actual process. The GA_PDA is understood as a software agent running in a PDA a Patient will carry constantly. The Patient will depend now on the GA_PDA to Have the Treatment Monitored. This *dependency* will be enforced by a *task* Monitor Treatment that refines this goal within the GA_PDA. Among other refinements, this *task* is decomposed by the *goal* Evaluate Treatment Compliance. Suggesting Changes in Drug use or Suggesting Changes in Diet or Exercise can satisfy this *goal*. By Reasoning About Personal Preferences to suggest changes in diet or exercise we would contribute positively (*help correlation link*) to the Patient's Flexibility *softgoal*. Furthermore, we need to respect the Autonomy that a Patient may have and in extension the Autonomy his GA_PDA may have. To Satisfice this Autonomy it is necessary to Respect Patient's Preferences for Communicating Problems to Parents. Further Refinement indicates that in order to comply with this *softgoal* the GA_PDA must check Patient's Preferences or Ask the Patient either the GA_PDA must send a notification to the parents or not. On the other hand, this alternative might contributes negatively (*hurt contribution link*) to the *softgoal* Parents Must be Always Informed About Problems in Treatment, which in turn refines the Safety *Softgoal*. An alternative way to satisfice this *softgoal* is to Keep the Physician Always informed About Frequent Problems in the Treatment. This way, the Safety *Softgoal* can be at least weakly satisfied without compromising the Autonomy *softgoal*.

Since the Patient now can tell the GA_PDA not to contact the parents about frequent occurrences of problems in treatment, now the Physician has to be Alerted Regarding Frequent Problems in Treatment so he can check whether or not changes may have to be made to the treatment. The GA_Physician software agent will carry this out and it is illustrated in Figure 7. The GA_Physician will be alerted by the GA_Home about any frequent problems in treatment. The GA_Home will in turn receive an alert from the GA_PDA about frequent problems together with instructions either to alert the parents or not. Another role for the GA_Physician will be to Help Physicians on Establishing a Treatment. This would have positive contributions (*help contribution links*) to the Physician *softgoals* Be Updated with Recent Drugs and Be updated with Recent Treatments. Those two *softgoals* that were previously considered weakly denied will be now reevaluated to weakly satisficed. In its turn the *softgoal* Quality is now evaluated as weakly satisficed.

Other alternatives could have been addressed but we limit ourselves to those two because of space constraint. The same way, the GA_Home would have to be refined and evaluated.

During the late requirements phase, these new software agents will be further detailed and analyzed. Non-functional requirements such as security, performance, availability and privacy will be considered and evaluated to their impact in the software design. Different architectures might be evaluated trying to determine which one would better suit to the particular characteristics of each agent.

4. Conclusions and Future Work

Recently, many different approaches have proposed solutions for analyzing and designing multi-agent systems [19][20]. This work extends the agent-oriented approach to requirements engineering, particularly at the early stages. The methodology aims to address some of the special challenges arising from the intentional and social dimensions of complex organizational environments. Agent characteristics such as autonomy, sociality, and intentionality are used to address the needs arising from large-scale software systems which may involve large number of stakeholders, playing different roles, with different goals and viewpoints.

The methodology presented in this paper is part of the Tropos Project [2]. The Tropos Project recognizes that existing methodologies such as object orientation and structured analysis have been motivated from programming languages rather than from characteristics of the world, and hence do not cope well with today's ever-changing world.

In our case studies, the methodology has been instrumental in guiding the elicitation and modelling activities. The agent-oriented focus helped surface many conflicting views, divergent goals, different ways of accomplishing goals, as well as issues of politics and power. Conventional techniques would not offer any support for dealing with these issues, even if the requirements engineer can recognize some of these complexities based on personal experience or intuition.

Studies of systems in use have long recognized how social factors can lead to system success or failure (e.g., [8]). An agent-oriented methodology offers a more explicit and systematic treatment of the intentional and social dimensions as an integral part of the system development process. An agent-oriented requirements methodology could also lead naturally into systems that employ agent-based software technology [2] [16]. Agent-centred modelling abstractions originating from the requirements phase can be used to preserve notions of autonomy and intentionality as the development process progresses to architecture to design to implementation, resulting in systems that are more flexible and robust, with an awareness of their social surroundings.

Large-scale software frequently suffers in later phases from not being able to model and justify many decisions that were made during the elicitation process. Since the methodology captures intentions it is possible to keep a rationale showing why one alternative was chosen over another.

We have been conducting case studies at three major hospitals in the Toronto area. We worked with stakeholders to explore how software could help overcome the many problems in the discharge process in these hospitals. We were able to propose a different discharge planning process supported by a new software agent with encouraging results. We are now investigating whether the same solution can be extended to the other two hospitals. Since the three hospitals deal with different types of patients and assessments and have different balance of powers among physicians, nurses and social workers, the answers are not immediately apparent.

Although results were positive, the case studies also revealed a number of limitations and ideas for future work. Complementary use of other techniques (such as scenarios and viewpoint management) in a more systematic way should be explored. Scenarios can be used to help identify missing actors or relationships. Viewpoint

techniques can be used to uncover how different actors perceive processes and relationships, and also how they would achieve goals. In our case studies, we found more than four different ways in which a patient is discharged. They achieve their higher-level goals in different ways some better than the others. Exploring the different roles involved in the domain enhances the chances of finding inconsistencies in the model as well as different ways of achieving a goal.

Better coupling between the lexicon tool and the ***i**** modelling tool can provide support for consistency checks and traceability. Semi-automated heuristic support for recognizing intentional concepts from natural language descriptions would be very desirable.

The methodology should also provide more guidance in the use of modelling constructs, e.g., typical uses of the agents, roles, and positions. Lighter weight versions of the methodology would also be useful for less demanding types of organizational environments.

We intend to expand the methodology to deal with the late phase of requirements engineering proposing a systematic approach to further refine requirements for software agents including the reasoning about possible software architectures. Links to other methodologies should also be the subject of future work. Some initial work on linking ***i**** to UML, particularly to Use Cases and Class diagrams, have recently been done [11][2].

5. References

[1] Boehm, Barry e In, Hoh. "*Identifying Quality-Requirement Conflicts"*. IEEE Software, March 1996, pp. 25-35

[2] Castro, J., M. Kolp, J. Mylopoulos. "A Requirements-Driven Development Methodology", 13th International Conference on Advanced Information Systems Engineering CAiSE 01, Interlaken, Switzerland, June 4-8, 2001.

[3] Chung, L., B.A. Nixon, E. Yu, J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[4] Cysneiros, L.M. and Leite, J.C.S.P. "Using UML to Reflect Non-Functional Requirements", Proc. of the CASCON 2001, Toronto, Nov 2001.

[5] DeMarco, T. *Structured Analysis and System Specification.* New York, Yourdon 1978.

[6] D'Souza, D.F and A.C. Will. *Objects, Components and Frameworks With UML: The Catalysis Approach*. Addison-Wesley 1999.

[7] Goguen, J. and C. Linde. "Techniques for Requirements Elicitation*" First International Symposium on Requirements Engineering*, IEEE Computer Society Press, pp152-164, 1993.

[8] Kling, Rob (ed.). *Computerization and Controversy: Value Conflicts and Social Choices.* 2nd. Edition. San Diego: Academic Press. 1996.

[9] Leite J.C.S.P. and A.P.M. Franco "A Strategy for Conceptual Model Acquisition " in Proceedings of the *First IEEE International Symposium on Requirements Engineering, SanDiego, Ca, IEEE Computer Society Press,* pp 243-246, 1993.

[10] Nuseibeh,B.A. and Easterbrook,S.M. "Requirements Engineering: A Roadmap", In A. C. W. Finkelstein (ed) "*The Future of Software Engineering* ". (Companion volume to the Proc. of the 22nd Int. Conf. on Software Engineering, ICSE00) IEEE Computer Society Press.

[11] Santander, V. and Castro, J. "Deriving Use Cases from Organizational Modelling" IEEE Joint International Requirements Engineering Conf. pp:32-39 Essen, Germany, Sept 2002.

[12] Ross, D. "Structured Analysis: A language for Communicating Ideas" *IEEE Trans. on Software Eng, 3(1), pp 16-34,* Jan. 1977.

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

[14] Szolovits, P., Doyle, J., Long, W.J. "Guardian Angel:Patient-Centered Health Information Systems" Technical Report MIT/LCS/TR-604, http://www.ga.org/ga/manifesto/GAtr.html

[15] Yu, E. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering" *in Proc. Of the 3rd IEEE Int. Symp. on Requirements Engineering*, pp:226-235, 1997.

[16] Yu, E. "Agent-Oriented Modelling: Software Versus the World". Agent-Oriented Software Engineering AOSE-2001 Workshop Proceedings. LNCS 2222.

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

[18] Wirfs-Brock R., B. Wilkerson and L. Wiener. *Designing Object-Oriented Software*, Prentice Hall 1990.

[19] The International Workshop series on Agent-Oriented Information Systems, http://www.aois.org/

[20] The International Workshop Series on Agent-Oriented Software Engineering, http://www.csc.liv.ac.uk/~mjw/aose/