Agent-Oriented Modelling: Software Versus the World

Eric Yu
Faculty of Information Studies
University of Toronto, Toronto, Canada M5S 3G6
yu@fis.utoronto.ca

Abstract. Agent orientation is currently pursued primarily as a software paradigm. Software with characteristics such as autonomy, sociality, reactivity and pro-activity, and communicative and cooperative abilities are expected to offer greater functionality and higher quality, in comparison to earlier paradigms such as object orientation. Agent models and languages are thus intended as abstractions of computational behaviour, eventually to be realized in software programs. However, for the successful application of any software technology, the software system must be understood and analyzed in the context of its environment in the world. This paper argues for a notion of agent suitable for modelling the strategic relationships among agents in the world, so that users and stakeholders can reason about the implications of alternate technology solutions and social structures, thus to better decide on solutions that address their strategic interests and needs. The discussion draws on recent work in requirements engineering and agent-oriented methodologies. A small example from telemedicine is used to illustrate.

1 Introduction

Agent orientation is emerging as a powerful new paradigm for computing. It offers a higher-level abstraction than the earlier paradigm of object orientation. Software agents have autonomy and are social; they communicate, coordinate, and cooperate with each other to achieve goals [3, 26, 50]. As agent software technology is maturing and entering into the mainstream, methods and techniques are urgently needed to guide system development in a production setting. Agent-oriented software engineering has thus become one of the most active areas in agents research (see, e.g., [9, 48]). For each application system, one needs to address the full range of software engineering issues – requirements, design, construction, validation and verification, deployment, maintenance, evolution, legacy, reuse, etc. – over its entire product life cycle.

Requirements engineering is an especially demanding, yet critical, task for a new technology such as agent-based software technology. Some adopters will have high expectations of the new capabilities, while others may be wary of potential pitfalls. Yet, most users will be unfamiliar with the new technology and unclear about its implications and consequences. Consider the healthcare domain. The potentials for applying agent technology, along with other kinds of information technology, are far-reaching. One can easily envisage software agents enhancing the information and communication infrastructure by offering better semantic interoperability, local autonomy, dynamic management of resources, flexible and robust exception handling, etc. Yet, it is by no means straightforward to go from idealized visions to viable systems in actual contexts. In real-life application settings, there are many competing requirements, and different interests and concerns from many stakeholders. As with any software technology, each stakeholder may be asking:

- What do I want the software to do for me? What can it do for me? Would I be better off to do the job myself, or to delegate to another human, or to another (type of) system?
- Can the software be trusted? Is it reliable? Will I have privacy?
- How do I know it will work? What if some function fails – what aspects of my work will be jeopardized? How do I mitigate those risks?
- What knowledge and information does it depend on? Where do they come from? How do I know they will be accurate and up-to-date, and effective? Will my skills and expertise continue to be valued?
- Will my job be easier? tougher? Will my position be threatened? How will my relationships with other people (and systems) change?

With agent technology, these issues and questions are accentuated by its greater reliance on codified knowledge, by its supposed flexibility and adaptivity, and by its autonomy (and thus possibly reduced perspicuity). Given their "intelligence" capabilities, agent systems can be expected to do more decision-making and problem solving. How does one decide what responsibilities to turn over to agent systems? Agent technology (including multi-agent systems) opens up many more opportunities and choices. At the same time, the task of exploring and analyzing the space of possibilities and their consequences has become much more complex.

The Requirements Engineering (RE) area in Software Engineering has been developing techniques for dealing with these issues. It has been recognized that poor requirements had led to many system failures and discontinued projects (see, e.g., [43]). Requirements engineering focuses on clarifying and defining the relationship between intended systems and the world. The introduction of a new system (or modification of an existing one) amounts to a redistribution of tasks and responsibilities among agents in the world – humans, hardware, software, organizational units, etc. Requirements engineering is therefore more than the specification of behaviour, because the ultimate criteria for system success is that stakeholders' goals are achieved and their concerns addressed.

Recent work in requirements engineering has thus adopted an agent-oriented perspective. The notion of agent in Requirements Engineering, however, is about agents in the world, most of which the software developer has no control over. The purpose of introducing an agent abstraction (or any other abstraction) for requirements modelling is to support elicitation, exploration, and analysis of the systems-in-the-world, possible alternatives in how they relate to each other, and the pros and cons of the alternatives. The requirements engineer needs to help users and stakeholders articulate their needs and concerns, explore alternatives, and understand implications. Thus, while agents-as-software and agents-in-the-world may share conceptual features, their respective abstractions are introduced for different reasons and serve different purposes. Characteristics such as intentionality, autonomy, and sociality have different connotations and consequences when treated as attributes of software than as attributes of agents in the world. In proposing or choosing an agent abstraction, different criteria and tradeoffs need to be considered.

In this paper, we examine the notion of agent as applied to the modelling of agents-in-the-world. In Section 3, we offer an outline of the ***i**** framework as an example of an agent-oriented modelling framework. Section 4 reviews the main contrasts between the notion of strategic agent-in-the-world, versus that of agent-as-software. In section 5, we discuss recent related work in requirements engineering and agent-oriented methodologies. In Section 6, we suggest a broader conception of AOSE not exclusive to agent-oriented software, and argue that the strategic view of agents-in-the-world should guide the entire software engineering process.

2 From Modelling Software to Modelling the World

Most agent models and languages are intended as abstractions of computational behaviour, eventually to be realized as software programs. Such models are needed for specifying and for constructing the software. Different kinds of models are needed for different stages and aspects of software engineering. As agent technology is maturing, attention is turning to the development of a full set of complementary models, notations, and methods to cover the entire software lifecycle [9, 48].

In Requirements Engineering, the need to model the world has long been recognized, as requirements are about defining the relationship between a software system and its environment. The major activities in requirements engineering include domain analysis, elicitation, negotiation & agreement, specification, communication, documentation, and evolution [47, 34]. Modelling and analysis techniques have been devised to assist in these tasks.

The Structured Analysis techniques first popularized the use of systematic approaches for expressing and analyzing requirements. The Structured Analysis and Design Technique (SADT) focused on the modelling of activities and data (inputs, outputs, and control flows among activities), and their hierarchical decomposition [40]. Dataflow diagrams include information stores, as well as external sources and sinks, thus demarcating a system's interfaces with its environment [12]. Complex descriptions were reduced into structured sets of diagrams based on a small number of ontological concepts, thus allowing for some basic analysis. For example, one could check completeness and consistency by matching input and output flows. Later on, these tasks were supported by CASE tools, although support is limited by the degree of formality in the models. These techniques continue to be widely used.

As the size of the models grew, and the need for reuse became recognized, structuring mechanisms were introduced to manage the knowledge in the models and to deal with complexity. For example, RML [21] provided for classification, generalization, aggregation, and time. To strengthen analysis, various approaches to formalization were introduced, accompanied by appropriate ontological enhancements. For example, RML offered assertions in addition to activities and entities, and provided semantics based on translation to first-order logic. Temporal, dynamic, deontic and other logics have also been introduced for requirements modelling [34]. Many of these features subsequently found their way into object-oriented modelling (e.g., UML [41]), which packages static and dynamic ontologies together into one behavioural unit. However, the analysis done with these models continue to be about the behaviour and interactions. There are no intentional concepts or considerations of strategic alternatives.

The Composite Systems Design approach [16, 15] first identified the need to view systems and their embedding environments in terms of agents that have choice. An agent's decisions and actions can place limits on other agent's freedom of action. In the KAOS framework [10], global goals are reduced through and/or refinement until they can be assigned as responsibilities to agents. These become requirements for systems to be built, or assumptions about agents existing in the environment. Goal modelling has been incorporated into a number of RE frameworks [57]. They provide incremental elicitation of requirements (e.g., [38]). They support the repeated use of "why, how, and how else" questions in the constructions of means-ends hierarchies, to understand motivations, intents, and rationales [52]. They reveal conflicts and help identify potential resolutions [39]. Quality goals constrain choices in a design space and can be used to guide the design process [8].

The introduction of goals into the ontology of requirements models represented a significant shift. Previously, the world to be modelled consisted of entities and activities and their variants. The newer ontologies attributed goals to agents in the world. In other words, to do requirements engineering, it is not enough to attend to the static and dynamic aspects, one also need to acknowledge intentionality in the world.

While recent research in requirements has given considerable attention to goals, the concept of agent has not been developed to the same extent. In particular, few RE frameworks have elaborated on or exploited concepts of agent autonomy, sociality, etc. The logical next step for RE is to go from goal-oriented requirements engineering to full-fledged agent-oriented requirements engineering, to acknowledge the social as well as the intentional [54, 33]. The need for this step is apparent as one considers the changing nature of systems and their environments. In the past, systems and their environments were much more stable and well delineated. Systems tended to be developed in isolation in relation to other systems. So the simplifying assumptions were that global goals could be identified, and that differences could be resolved to achieve agreement across the system.

Today, most systems are extensively networked and distributed, operating under multiple jurisdictions each with their own mandates and prerogatives. Stakeholders want local autonomy but cooperate on specific ventures. They depend on each other, and on each other's systems in multiply-connected ways. They have limited knowledge about each other, and have limited influence and control over each other. The traditional mechanistic worldview needs to give way to a more sophisticated social worldview [55].

In the next section, we outline a modelling framework in which agents play a central ontological role. The framework begins to address the more complex relationships and issues that arise in Requirements Engineering. Agents-in-the-world are taken to be intentional and semi-autonomous. They associate with each other in complex social relationships. Their identities and boundaries are contingent. They reflect upon their relationships in the world and strategize about alternate relationships in order to further their interests.

It must be recognized that the framework represents only one possible approach. In adopting a richer ontology, one gains in expressiveness and analytical power. On the other hand, it places greater demands on elicitation and validation. So there are significant trade-offs that need to be considered in the context of an overall development methodology.

3 A Framework for Modelling Agents-in-the-World

Consider a home care scenario in which a patient receives remote monitoring and telemedicine services from one or more healthcare service providers – hospitals, physicians, nurses, pharmacies, laboratories, clinics, emergency centres, consultants, etc., allied to varying degrees but sometimes also in competition.[1] Such arrangements can potentially improve quality of care and reduce overall healthcare costs, while allowing patients to lead more normal lives at home. Agent technology can be used to achieve greater functionality, robustness, and flexibility in such systems, for example, by incorporating knowledge-based decision support and knowledge discovery, by offering context-aware initiatives and failure recovery, by enabling dynamic resource discovery, negotiation, and mediation, or by facilitating collaboration among individuals and groups through multimedia and logistics support, and cooperation among disparate systems. Patients get more customized care while healthcare professionals are relieved of the more mundane aspects of their tasks.

But how does one decide what functionalities the systems should have? Who should these systems be accountable to? How should responsibilities be divided among them, and why? Do the stakeholders have common goals? Can the systems function despite ongoing differences and competing interests? Clearly these questions would results in different answers for each setting, depending on the context. In each setting, there could be numerous options to consider. Some may appear workable initially, but may turn out to be, upon further analysis, technical infeasible, or unacceptable to certain stakeholders. During requirements engineering, it is important for all stakeholders, customers, users, system developers, and analysts to understand each other's interests and concerns, to jointly explore options, and to appreciate the implications of alternative decisions about the systems to be constructed.

In the past, notations and methods in software development have focused more on the specification of systems after these decisions have been made. Few of the commonly used notations, e.g., UML, provide explicit support for expressing, analyzing, and supporting decisions about these issues.

Today, systems and their surrounding context in the world are constantly changing. Aside from rapid technological innovations, systems need to respond to frequent changes in organizational structures, business models, market dynamics, legal and regulatory structures, public sentiments and cultural shifts. We need systematic frameworks – models, methods, and tools – to support the discovery of requirements, analysis of their implications, and the exploration of alternatives.

The ***i**** framework [53, 52] is used to model and analyze relationships among strategic actors in a social network, such as human organizations and other forms of social structures. Actors are semi-autonomous units whose behaviours are not fully

---

[1] This home care setting is loosely based on [23].

controllable or predictable, but are regulated by social relationships. Most crucially, actors depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. By depending on someone else, an actor may achieve goals that would otherwise be unachievable. However, a dependency may bring along vulnerabilities since it can fail despite social conventions such as commitments. The explicit representation of goals allows the exploration of alternatives through means-ends reasoning. A concept of softgoal based on the notion of satisficing is used to provide a flexible interactive style of reasoning.

Note that in the context of modelling the world, unlike the modelling of software agents for the purpose of construction, qualities such as autonomy and sociality are being externally ascribed to some elements in the world for the purpose of description and analysis. Some selected elements depicted in the models may end up being implemented as software agents, others may materialize as more conventional software, while many of them are, and will remain mostly human wetware. The implementational construction of the actors is irrelevant to this level of modelling of the world. These considerations will be further discussed in Section 4.

The ***i**** modelling framework consists of two types of models – the Strategic Dependency (SD) model and the Strategic Rationale (SR) model.

3.1 Modelling intentional relationships among strategic actors – the Strategic Dependency model

The Strategic Dependency (SD) model is a graph, where each node represents an actor, and each link between two actors indicates that one actor depends on the other for something in order that the former may attain some goal. We call the depending actor the depender, and the actor who is depended upon the dependee. The object around which the dependency relationship centres is called the dependum. An actor is an active entity that carries out actions to achieve goals by exercising its knowhow. In the SD model, the internal goals, knowhow, and resources of an actor are not explicitly modelled. The focus is on external relationships among actors.

Figure 1 shows a Strategic Dependency model of a (much simplified) telemedicine setting. A Patient depends on a Healthcare Provider to have Sickness Treated. The latter in turn depends on the patient to Follow a Treatment Plan. As the patient would like to integrate the treatment into other activities, she wants the treatment plan to be Flexible. The healthcare provider partly addresses this by monitoring vital signs remotely, with the help of equipment on the patient site (Monitoring Agent), and a host system (Monitoring System) that oversees a number of patients.

The SD model expresses what actors want from each other, thus identifying a network of dependencies. The intentional dependencies, in terms of wants and abilities to meet those wants, are expressed at a high level, so that details about information and control flows and protocols are deferred. Even at this high level, many issues are already apparent. Healthcare Provider enables Patient to achieve the Sickness Treated goal that the latter may not be able to achieve on her own. In taking advantage of this opportunity, the depender becomes vulnerable to the dependency. The model assists them in deciding whether their dependencies are acceptable, or that they should seek alternate arrangements.

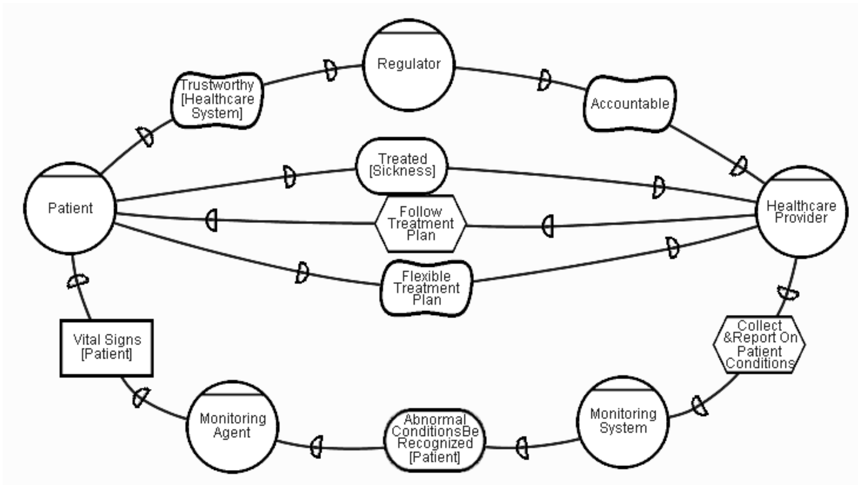

Fig. 1. A Strategic Dependency model

Four types of dependencies are distinguished for indicating the nature of the freedom and control in the relationship between two actors regarding a dependum. In a *goal dependency*, the depender depends on the dependee to bring about a certain state of affairs in the world. The dependum is expressed as an assertional statement. The dependee is free to, and is expected to, make whatever decisions are necessary to achieve the goal (the dependum). The depender does not care how the dependee goes about achieving the goal. For example, Patient depends on Healthcare Provider to have Sickness Be Treated. It is up to the Provider to choose how to treat the sickness, as long as the goal is achieved.

In a *task dependency*, the depender depends on the dependee to carry out an activity. The dependum names a task which specifies how the task is to be performed, but not why. The depender has already made decisions about how the task is to be performed. Physician depends on Patient to Follow Treatment Plan, described in terms of activities and sub-activities, possibly with constraints among them, such as temporal precedence. Note that a task description in ***i**** is not meant to be a complete specification of the steps required to execute the task. It is a constraint imposed by the depender on the dependee. The dependee still has freedom of action within those constraints.

In a *resource dependency*, the depender depends on the dependee for the availability of an entity (physical or informational). By establishing this dependency, the depender gains the ability to use this entity as a resource. A resource is the finished product of some deliberation-action process. In a resource dependency, it is assumed that there are no open issues to be addressed or decisions to be made. For example, Vital Signs from the patient is treated as a resource, as it is not considered problematic to obtain.

In a *softgoal dependency*, a depender depends on the dependee to perform some task that meets a softgoal. A softgoal is similar to a goal except that the criteria of success are not sharply defined a priori. The meaning of the softgoal is elaborated in terms of the methods that are chosen in the course of pursuing the goal. The depender decides what constitutes satisfactory attainment ("satisficing" [42]) of the goal, but does so with the benefit of the dependee's knowhow. Whether a treatment plan is considered to be sufficiently Flexible is judged by the Patient, with the Healthcare Provider offering alternate methods for achieving flexibility. Similary, Trustworthiness of the healthcare system, and Accountability of the healthcare provider are treated as softgoals, since there are no clear-cut criteria for their satisfaction.

The model also provides for three degrees of strength of dependency: open (uncommitted), committed, and critical. These apply independently on each side of a dependency.

Actors can assess the desirability of alternate configurations of relationships with other actors according to what they consider to be significant to them. The viability of a dependency can be analyzed in terms of enforceability (Does the other actor depend in return on me for something, directly or indirectly?), assurance (Are there other dependencies on that actor that would reinforce my confidence in the success of that dependency?), and insurance (Do I have back-ups or second sources in case of failure?). Strategic dependencies can be analyzed in terms of loop and node patterns in the graph.

The generic concept of strategic *actor* outlined above can be further differentiated into the concepts of role, position, and agent [56]. A *role* is an abstract collection of coherent abilities and expectations. A *position* is a collection of roles that are typically occupied by one agent. An *agent* is an actor that has concrete manifestations such as human, hardware, or software, or combinations thereof. Agents, roles, and positions may also be composed into aggregate actors.

3.2 Modelling the reasoning behind strategic relationships – the Strategic Rationale model

Whereas the Strategic Dependency model focuses on relationships between actors, the Strategic Rationale (SR) model provides support for modelling the reasoning of each actor about its intentional relationships. The SR model is a graph whose nodes are goals, tasks, resources, and softgoals. These may be connected by means-ends links or task-decomposition links. A goal may be associated, through means-ends links, with multiple, alternative ways for achieving it, usually represented as tasks. The means-ends links for softgoals, however, require more differentiation because there can be various types of contributions leading to a judgement of whether the softgoal is sufficiently met ("satisficing"). These include Make, Break, Help, Hurt, Positive, Negative, And, Or, Unknown, and Equal [8]. Task-decomposition links provide hierarchical decomposition of tasks into subtasks, subgoals, resources, and softgoals that make up the task.

Figure 2 is an SR model showing some of the reasoning behind one possible telemedicine arrangement. It has been argued that current healthcare systems are too provider-centred, in that patients have little control over the information collected about them, and cannot participate effectively in their own care.[2]

[2] The patient-centred scenarios draw on those of [45] and [30].

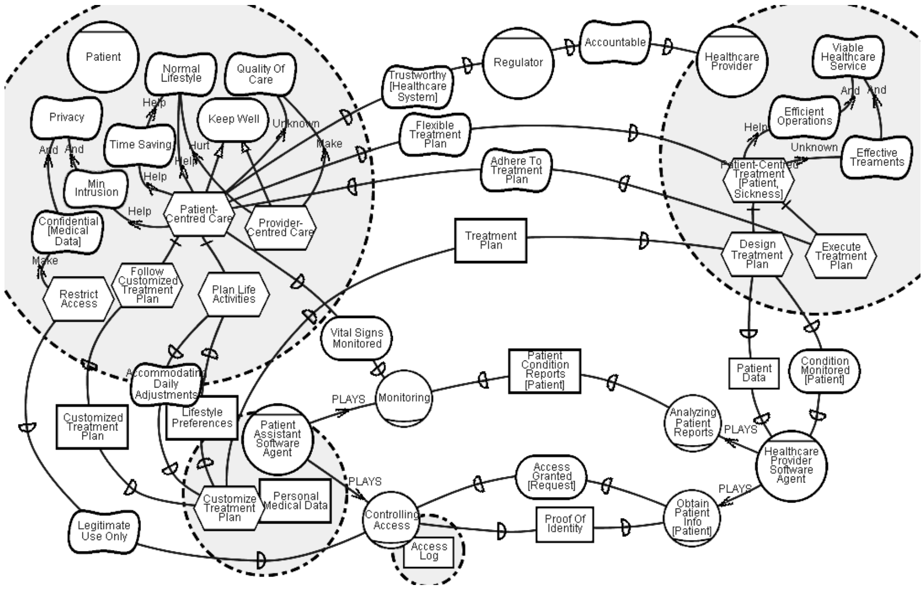

Fig. 2. A Strategic Rationale model showing some reasoning behind patient-centred care

One way to achieve patient-centred care is to have the full medical records and history of the patient controlled by the patient. A software agent acting in the interest of the patient would grant access to healthcare providers for legitimate use. This arrangement is in contrast to the current practice in which each provider generates and keeps their own records, resulting in fragmented, partial views, delays and duplication (e.g., the same lab tests repeated at multiple sites). The integrated personal medical data would also allow the intelligent assistant to customize treatment plans to suit the specific needs and the lifestyle of the patient. The healthcare provider monitors the progress of the patient through her own software agent assistants.

The SR model for the Patient in Figure 2 shows that the patient has the goal of Keeping Well, but is also concerned about Privacy, Quality Of Care, and maintaining a Normal Lifestyle. The SR modelling constructs allow the systematic refinement of these goals to explore ways for achieving them. According to the model, Privacy is achieved if the medical data is kept Confidential, and if Intrusion Is Minimized (And). Confidentiality is sufficiently addressed (Make) if Access Is Restricted. The goal of Keeping Well can be accomplished with Patient-Centred Care or with Provider-Centred Care (means-ends links). Patient-Centred Care involves the subtasks of Follow Customized Treatment Plan and Plan Life Activities. These subtasks have dependencies with the Patient Assistant Software Agent.

The example model is greatly simplified but provides some hints on the types of reasoning to be supported. These include the raising of issues, the identification and exploration of alternatives, recognition of correlated issues (good and bad side-effects), and the settling of issues. For example, while Patient-Centred Care contributes positively to Privacy and Normal Lifestyle, its contribution to Quality Of Care is Unknown. This suggests further elaboration and refinement of the Quality Of Care softgoal so that the nature of the contributions can be better assessed. Elaboration of this and other goals may help discover other kinds of provider-centred and patient-centred care, each of which may have different contributions to the various goals.

We have presented ***i**** in terms of a graphical representation. ***i**** modelling is implemented on top of the Telos conceptual modelling language [31], which offers knowledge structuring mechanisms (classification, generalization, aggregation, attribution, time). Generic knowledge codified in terms of methods and rules provide semi-automatic support from a knowledge base. A prototype tool has been developed to support ***i**** modelling. Further analysis support is being developed in the Tropos project [32].

4 Agents-in-the-World versus Agents-as-Software

Having reviewed ***i**** as an example framework for modelling agents-in-the-world, we now consider some of the key issues in designing such frameworks. These issues help clarify the distinctions between modelling agents in the world versus modelling agents as software entities. We consider the issues of autonomy, intentionality, sociality, identity and boundaries, strategic reflection, and rational self-interest. While most of these issues have their counterparts in agents-as-software, their significance for modelling agents-in-the-world are quite different.

4.1 Autonomy

Traditional requirements analysis techniques rely heavily on the modelling of processes or interactions. Through activity diagrams, event sequence charts, etc., one describes or prescribes what would or should happen under various known conditions. Real-life practice, however, often departs from these idealizations [44] and frequently require workarounds [19]. There are many aspects of the world over which one has little control or knowledge, so it is hard to anticipate all contingencies and be able to know in advance what responses are appropriate.

Thus, in introducing autonomy into a model of agents-in-the-world, we are adopting a less simplistic view of the world, so as to take uncertainties into account when judging the viability of proposed alternatives, such as different ways for achieving patient-centred care using software agents. Agents-in-the-world need to be aware of uncertainties around them. At the same time, they themselves are sources of uncertainty in the world.

In devising a modelling scheme that acknowledges agent autonomy, the challenge is to be able to describe or prescribe agent behaviour without precluding openness and uncertainties. In ***i****, actors are assumed to be autonomous in the sense that the analyst should not rule out any behaviour. An actor's dependencies and strategic interests provide hints on the actor's behaviour, but do not provide guarantees. Thus, one would be well advised to adopt mechanisms for mitigating risks, based on an analysis of vulnerabilities, e.g., backup systems and procedures in case of failure in the patient monitoring system. The dependency types in ***i**** are used to differentiate among the types of freedoms that actors have with regard to some specific aspect of the world, as identified by the dependum.

For agents-as-software, autonomy refers to the ability of the software to act independently without direct intervention from humans or other agents. It is a desired property that must be consciously created in the software. It is a property only achievable with recent advances in software and artificial intelligence technology. For agents-in-the-world, autonomy is an inherent property, but it has been ignored in the past for simplicity of modelling. Now we want it back because we want to face up to these more challenging aspects of the world. For software agents, greater autonomy implies more powerful software, which are likely to be more challenging to design and implement. For modelling the world, allowing greater autonomy in the agent model means one would like to analyze the implications of greater uncertainties and variability in the world.

4.2 Intentionality

Conventional requirements analysis (e.g., as supported by UML) assumes complete knowledge and fully specifies behaviour, so there is little need for intentional concepts. To account for uncertainties and openness in the world, however, intentional concepts such as goals and beliefs can be very useful. In modelling agents-in-the-world, we ascribe intentionality to them so as to characterize alternate realities in the world. Some of these alternate realities are desirable, but an agent may not know how to get there, or may not want to fix the path for getting there to allow for flexibility. Intentional concepts thus allow agents to be described without detailing specific actions in terms of processes and steps. Explicit representation of goals allows motivations and rationales to be expressed. They allow "why" questions to be raised and answered. Beliefs provide for the possibility that an agent can be wrong in its assumptions about the world, and mechanisms to support revisions to those assumptions.

For agents-as-software, intentionality is a property that is used to generate the behaviour of the agent. For example, there may be data structures and internal states that represent goals and beliefs in the software. For agents-in-the-world, we do not need to presuppose intentionality in their internal mechanisms. Multi-agent modelling allows different goals, beliefs, abilities, etc., to be attributed to different agents. An agent can be thought of as a locality for intentionality. Instead of having a single global collection of goals, belief, etc., these are allocated to separate agents. The agent concept provides a local scope for reconciling and making tradeoffs among competing intentionality, such as conflicting goals and inconsistent beliefs.

4.3 Sociality

Traditional systems analysis views systems and their environments mechanistically. They produce outputs from inputs, either as pre-defined processes or as reactive responses to control signals or events. Complexity and scalability is primarily dealt with by composition or decomposition, with the behaviour of the whole being determined by the behaviour of the parts together with compositional rules. When systems and their environments have autonomy, these assumptions no longer hold. Active autonomous entities in the world have their own initiatives, and are not necessarily compliant with external demands or desires, such as those from a system designer. Autonomous agents can choose to cooperate, or not, to varying degrees, and on their own terms. A social paradigm is needed to cover the much richer kinds of relationships that exist in such settings.

Social agents have reciprocal dependencies and expectations on each other. They tend to have multi-lateral relationships, rather than one-way relationships. Agent A can expect agent B to deliver on a commitment because B has goals and interests that A can help fulfil or meet. Reciprocity can be indirect, mediated via other agents. In general, social relationships exist as networks and patterns of relationships that involve multi-lateral dependencies. In mechanistic artificial systems, where one designer oversees interaction among parts, it is more common to see master-slave relationships that go one-way.

Social agents typically participate in multiple relationships, with a number of other agents, at the same time or at different times. In mechanistic systems as portrayed in most traditional models, relationships are narrowly focused around intended functions.

Conflicts among many of the relationships that an agent participates in are not easily resolvable. There may be conflicts or potential conflicts arising from the multiple relationships that an agent engages in. In traditional approaches, competing demands need to be reconciled in order for requirements to be defined, then frozen for system development and implementation. In a more fluid and open environment, the demands of various agents may keep changing and may not be fully knowable. Agents may also build new relationships with other agents and dissolve existing ones. The management of conflicts is an ongoing one. Therefore it becomes necessary to maintain an explicit representation of the competing interests and their conflicts.

Agent relationships form an unbounded network. There are no inherent limits on how far the impact of dependencies may propagate in a network of agents. In considering the impact of changes, one may ask: Who else would be affected? Who will benefit, who will be hurt? Who can help me improve my position? These questions may lead to the discovery of agents not previously considered.

Cooperation among agents cannot be taken for granted. The potential for successful cooperation may be assessed through the analysis of agents' goals and beliefs. Techniques are needed to support the analysis of various aspects of cooperation, including synergy and conflict among goals, how to discover shared goals, and how goals may change.

For software agents, sociality refers to properties that must be created in the software to enable them to exhibit richer behavioural patterns. For agents-in-the-world, sociality refers to the acknowledgement of the complex realities in the world. Instead of abstracting them away as in earlier modelling paradigm, we try to device modelling constructs and analysis techniques that encompass them.

4.4 Identity and Boundary

In a social world, identities and boundaries are often contingent and contentious. Many social or organizational problems arise from uncertainties or disputes about boundaries and identities. For example, software agents working on behalf of or in cooperation with healthcare workers need to deal with a complex array of organizational roles, positions, and professions, often with sensitive relationships among them. Requirements analysis needs to be able to deal with these, to arrive at viable systems.

Boundaries and identities change, usually as a result of ongoing social processes such as socialization, negotiation, and power shifts. Technical systems often introduce abrupt changes in boundaries and identities, as they reallocate responsibilities and powers. Agents-in-the-world are concerned about their boundaries, and may attempt to change them to their advantage. Boundaries may be based on concrete physical material criteria, or abstract concepts such as responsibilities. In ***i****, dependums serve as actor boundaries at an intentional level. The boundaries are movable as dependums can be brought "inside" an actor or moved "outside" along means-ends hierarchies in the Strategic Rationale model. The ***i**** constructs of role, position, and agent distinguish among abstract and concrete actors, and provide mappings across them.

In models for agents-as-software, issues of identity and boundary can be much simpler, if all the agents are within the control of a designer. They would be determined by design criteria such as functional specialty, coordination efficiency, robustness, flexibility, etc. However, if the agents in a multi-agent system are designed and controlled by different designers and operators, and are thus autonomous in the social (agents-in-the-world) sense, then the more complex social notions of identity could be applicable.

4.5 Strategic Reflectivity

Traditional requirements models are typically used to express one way – the intended way – in which the system will operate in the world. Even if a space of alternatives was explored in arriving at the requirements, there is little representational or reasoning support for navigating that space. With today's systems undergoing frequent changes, the need to support evolution and to avoid legacy problems is well recognized.

Reasoning about alternative arrangements of technical systems in the world is a reflective process. Agents need to refer to and compare alternate ways of performing tasks, rather than executing the tasks without question. The reflective process is strategic because agents want to determine which changes would better serve their strategic interests. For example, patients want healthcare technologies that improve the quality of care while protecting their privacy. Hospitals may want greater efficiency without increased dependence on high-cost professionals.

During requirements analysis, strategic reflection is carried out by the human stakeholders, assisted by requirements analysts. In software agents, this kind of strategic reflection can potentially be done at run-time by the software. This characteristic requires higher sophistication to be built into the software (see, e.g., [1]) and is not yet a common feature. Strategic reflection is, however, a fairly basic need at the requirements stage.

4.6 Rational Self-interest

Most languages for modelling and requirements analysis (e.g., UML) do not provide explicit support for rationales. Since their ontologies do not include autonomous agents-in-the-world, the rationales, even if made explicit, would likely be a rationalization of the many contributions that led to the eventual requirements for a new system. In treating systems and environments as a multi-agent world, we try to explicate the preferences and decisions of each stakeholder in terms of rational self-interest. Each agent selects those options that best serve its interests. This assumption provides a convenient idealization for characterizing agents whose behaviour are otherwise unpredictable. Note that rational self-interest does not imply selfishness, as an agent can have altruistic goals.

The modeller attributes rationality and coherence to agents-in-the-world in order to draw inferences about their behaviour. However, the inferences are limited by incomplete and imperfect knowledge. The rationality is bounded and partial. The agent construct can be viewed as a scoping mechanism for delineating the exercising of rationality within a limited local scope.

In contrast, for software agents, rationality is a regime for governing the behaviour of the software according to internal states of goals and beliefs. Again, it is a characteristic that needs to be explicitly built into the construction of a software agent.

4.7 Summary

To summarize, agent concepts are useful both for software construction and for modelling the world. However, abstractions for agents-as-software and agents-in-the-world came about with different motivations, premises, and objectives, and thus can differ in ontology.

For software agents, the objective is to create a new software paradigm that would make software more powerful, more robust, and flexible. The realization of software agent characteristics requires greater sophistication in the implementation technology, which are ideally hidden under the agent abstraction.

In contrast, in devising some concept of agent for modelling the world, we recognize that the world already exists in its full complexity. Earlier modelling paradigms have adopted abstractions that removed too much of that complexity, resulting in ontologies that are too impoverished for dealing with today's problems. The agent abstraction is used to being back some of that complexity and richness to support appropriate kinds of modelling and analysis.

In either case, there is choice in what agent abstraction to adopt. For software agents, we want a concept of agent that fully embodies the behaviour to be generated. We need to consider the feasibility of implementation, and the difficulty of verifying implementation against the specification. For modelling agents-in-the-world, we want rich enough description of the world (expressiveness) to allow us to make the distinctions that we want, leading to analyses that matter in stakeholders' decision making. We do not want more detail than we can use, since there are costs in elicitation and validation, and potential for errors.

5 Related Work

Most of the current work in Agent-Oriented Software Engineering (AOSE) originated from the programming and AI/DAI systems construction perspective. As the technology infrastructure matures, attention is increasingly being paid to software engineering and application methodology issues. The focus therefore continues to have a strong systems construction flavour, with a gradual broadening to encompass contextual activities such as requirements engineering.

The predominant notion of agent in the current AOSE literature is therefore that of agent-as-software. Methodological frameworks have focused mostly on the "analysis and design" stages (e.g., [51, 2, 6, 27]). Requirements are assumed to be given, at least as informal descriptions. The analysis stage constructs a model of the intended behaviour of the software system.

The importance of requirements is beginning to be recognized, with attention being paid to the embedding environment. However, they are typically specified in terms of behavioural interactions, as in conventional requirements approaches. The notion of agent employed is still that of agent-as-software. For example, notions of autonomy, intentionality, etc., are those associated with the software, not with agents-in-the-world outlined in Section 4. Alternatives during requirements analysis, as viewed by strategic agents-in-the-world, are not explicitly addressed.

Social and organizational concepts are applied to software agents, not to agents in the world (e.g., [58, 36, 11, 13, 35]). Selective aspects of sociality are built into the agent software, with the purpose of enhancing the capabilities of the software, as opposed to the richer analysis of the environment for the purpose of defining the right technical system to build.

When reflection is used, it is as a computational mechanism in software agents (e.g., [1]), not used by stakeholders to reflect on strategic implications of alternative arrangements of technical systems in their environment.

In Requirements Engineering, agents have served as a modelling construct without assuming the use of agent software as the implementation technology. The concept of agent has been elaborated to varying degrees. For example, the EKD methodology [5] contains many of the concepts needed for agent-oriented modelling, but does not explicitly deal with issues of agent autonomy and sociality. Agents appear in one of six interconnected submodels: the Goal model, the Concepts model, the Business Rules model, the Business Process model, the Actors and Resources model, and the Technical Components and Requirements model. The KAOS approach [47] (also mentioned in Section 2) offers a detailed formal framework for eliciting and refining goals until they are reduced to operations that can be assigned to agents. The openness and autonomy of agent actions is not considered when generating or evaluating alternatives. Agents interact with each other non-intentionally, so they do not have rich social relationships. Both EKD and KAOS can be said to be more goal-oriented than agent-oriented.

Action-Workflow is a notation and method for modelling cooperative work [29]. The basic unit of modelling is a workflow between a customer and a performer. The customer-performer relationship is characterized in terms of a four-phased loop, representing the stages of proposing, agreeing, performing, and accepting. Each phase involves different types of communication acts which can be analyzed using Speech Acts theory. This framework has a stronger orientation to deal with the social nature of agents, especially their reliance on commitments and the potential for breakdowns. Intentional structures such as goals or means-ends relationships are not explicitly represented, so there is no support for reflection or shifting boundaries of responsibilities.

Many other techniques in Requirements Engineering bear close relations to agent modelling, e.g., managing multiple viewpoints [17], dealing with inconsistencies [20], supporting traceability [25] and negotiation [39], and scenario analysis [24].

While the ***i**** framework arguably goes farthest in addressing agent modelling issues in the spirit of this paper, many open issues remain, both in theoretical and practical areas. Recent work that have built on or extended ***i**** include the incorporation of temporal constraints to support simulation and verification [18, 49], development methodologies [14, 46], and multi-perspective modelling [37, 28].

The Tropos project [32, 7, 4] aims to develop a software development methodology that would carry the requirements ontology (based on ***i****) as far downstream as possible, to ensure that the resulting software would be requirements-driven. Agent orientation is assumed throughout all the development stages. Formal techniques are being developed to support analysis at various stages.

6 Engineering of Agent-Oriented Software vs. Agent-Oriented Engineering of Software

The predominant interpretation of the phrase "Agent-Oriented Software Engineering" is that of the engineering of software that uses the agent concept as the core computational abstraction. However, it is also possible to conceive of the use of agent concepts to support the software engineering process, without necessarily committing a priori to a software agent technology implementation. For the purpose of distinction, we could refer to the two conceptions of AOSE as EAOS and AOES respectively.

Agent-oriented techniques for requirements engineering, as exemplified by the ***i**** framework, suggests that agent concepts can be used profitably without prejudging the implementation technology. We have argued that issues of autonomy, intentionality, sociality, etc. are just as relevant in requirements engineering as in software construction, though in somewhat different senses.

A basic tenet in software engineering is to defer commitments on design and implementation decisions as much as possible, so as not to over-constrain those decisions unnecessarily. Conventional models and languages in software engineering – for requirements specification, architectural design, detailed design, programming, configuration, etc. – do not allow for the explicit representation of open decisions, freedoms and constraints, and argumentation about them. While each stage or activity in software engineering requires considerable deliberation and decision-making, the notations can only express and record the results of decision processes. Current notations provide hardly any support for the communication of intentional content among software engineers, e.g., design intents and rationales. Intermediate products in software engineering are passed on from one stage to another only after they are fully reduced to non-intentional representations, e.g., input/output relationships in architectural block diagrams.

Agent abstractions and models offer the expressiveness and flexibility that conventional notations lack. Today's increasingly fast-paced and fluid demands in software engineering suggests that agent abstractions could be useful for supporting software engineering processes in general. This is the premise behind the Tropos project [32, 7, 4]. Agent-based ontologies are used for representing requirements, architectures, and detailed designs. Intentional models involving goals and beliefs provide higher-level descriptions that allow suitable degrees of freedom. The ontologies that are appropriate are those for modelling agents-in-the-world. For the most part, the subject matter in software engineering activities are not (yet) software artefacts, but their precursors. While executable software would eventually emerge, many of the key engineering processes occur at the earlier stages where relationships among earlier design artefacts (e.g., architectural blocks or design modules) are worked out. The appropriate ontology is therefore not a computational ontology for machine execution, but a world ontology in which there are many human decision makers and stakeholders, exploiting opportunities, mitigating vulnerabilities, and choosing among alternatives according to strategic interests. The ***i**** framework is used as the starting point for the Tropos project.

Since software engineering work continues to rely heavily on human social processes, a full development of the AOES vision should include the many human players in a software engineering projects as full-fledged agents (or actors in ***i**** terminology). Human agents, roles, and positions would be interwoven with those representing the emerging artefacts. As the software development process unfolds, new actors and relationships would be created, existing ones evolve, others dissolve. The agents-in-the-world modelling paradigm allows a uniform representation of machine and human processes. This would support, for example, reasoning about whether an activity should be done at run-time or at development time, by human or by machine. These would be indicated as alternate boundaries among actors in ***i****. This conception of AOES is currently being explored [22].

Many software engineering challenges are not only technical, but social and organizational, e.g., reusability, maintainability, evolvability, comprehensibility, outsourcing, componentization, etc. A representation and engineering framework that provides full and equal treatment to technical artefacts as well as to human processes (including knowledge management and human capital considerations) can potentially offer a fuller account of software engineering, as well as more effective solutions.

While the general vision of AOES is independent of software implementation technology, the greatest benefit is obtained when the implementation does employ software agent technology. This would allow certain open decisions to be deferred to run-time to be executed by the agent software. Which ones to defer would be a frequent question that occurs throughout the development process. AOES modelling frameworks and tools should provide support for addressing such questions.

7 Conclusions

Agent orientation can contribute to software engineering in more ways than one. We have outlined a notion of agent from the viewpoint of requirements engineering, which focuses on the relationship between systems and their environments in the world. This notion of agent benefits from the development of the agent-as-software concept, but is distinct from it. We have outlined some major distinctions in terms of key agent properties such as autonomy and sociality. Because of the differences in context and objectives in different stages and aspects of software engineering, it is not surprising that differing agent abstractions have developed. However, as requirements engineering turns to face the new challenges raised by agent software technology, and as software agents acquire greater abilities to reason strategically about themselves and the world, one can expect closer links between conceptions of agents-as-software and agents-in-the-world. These are topics of ongoing research.

Acknowledgements. Financial support from the Natural Sciences and Engineering Research Council of Canada, Communications and Information Technology Ontario, and Mitel Corporation are gratefully acknowledged.

References

1. Barber, K.S., Han D.C., Liu, T.H.: Strategy Selection-based Meta-level Reasoning for of Multi-Agent Problem Solving. . In: Ciancarini, P., Wooldridge, M.J. (eds): Agent-Oriented Software Engineering: AOSE 2000. Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag. (2001) 269-284
2. Bauer, B., Müller, J.P., Odell, J.: An Extension of UML by Protocols for Multiagent Interaction. Proc. 4th Int. Conf. on Multi-Agent Systems. IEEE Computer Society. (2000) 207-214
3. Bradshaw, J. (ed.): Software Agents. AAAI Press (1997)
4. Bresciani, P., Perini, A., Giunchiglia, F., Giorgini, P., Mylopoulos, J.: A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. Proc. 5th Int. Conf. on Autonomous Agents, Montreal, Canada. (2001)
5. Bubenko, J., Brash, D., Stirna, J.: EKD User Guide. (1998). Available at ftp://ftp.dsv.su.se/users/js/ekd_user_guide.pdf
6. Caire, C., Garijo, F., Gomez, J., Pavon, J., Leal, F, Chainho, P, Kearney, P., Stark, J., Evans R., Massonet, P.: Agent Oriented Analysis Using MESSAGE/UML. In this volume.
7. Castro, J., Kolp, M., Mylopoulos, J.: A Requirements-Driven Development Methodology, 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), Interlaken, Switzerland. LNCS Vol. 2068 Springer-Verlag (2001) 108-123
8. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers. (2000)
9. Ciancarini, P., Wooldridge, M.J. (eds): Agent-Oriented Software Engineering: First Int. Workshop, AOSE 2000. Limerick Ireland, June 10, 2000. Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag. (2001)
10. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition, Science of Computer Programming. 20 (1-2): (1993) 3-50
11. Dastani, M., Jonker, C., Treur, J.: A Requirement Specification Language For Configuration Dynamics Of Multi-Agent Systems. In this volume.
12. DeMarco, T.: Structured Analysis and System Specification. New York: Yourdon, (1978)
13. Dignum, V., Weigand, H., Xu, L.: Agent Societies: Towards Frameworks-Based Design. In this volume.
14. Dubois, E., Yu, E., Petit, M.: From Early to Late Formal Requirements: a Process Control Case Study. Proc. 9th Int. Workshop on Software Specification and Design, Ise-Shima, Japan. IEEE Computer Society (1998) 34-42
15. Feather, M.S., Fickas, S.F., Helm, B.R.: Composite System Design: The Good News And The Bad News, Proceedings of Fourth Annual KBSE Conference, Syracuse. (1991) 16-25
16. Feather, M.S.: Language Support For The Specification And Development Of Composite Systems. ACM Trans.on Programming Languages and Systems , 9(2): (1987) 198-234
17. Finkelstein, A., Sommerville, I.: The Viewpoints FAQ: Editorial - Viewpoints in Requirements Engineering. IEE Software Engineering Journal, 11(1): (1996) 2-4
18. Gans, G., Jarke, M., Kethers, S., Lakemeyer, G.,.Ellrich, L., Funken, C., Meister, M.: Requirements Modeling for Organization Networks: A (Dis-)Trust-Based Approach. 5th IEEE Int. Symp. on Requirements Eng., Toronto, Canada. (2001)
19. Gasser, L.: Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics. Artificial Intelligence. 47(1-3): (1991) 107-138

20. Ghezzi, C., Nuseibeh, B.: Guest Editorial - Managing Inconsistency in Software Development. IEEE Transactions on Software Engineering 24(11): (1998) 906-907
21. Greenspan, S.: Requirements Modelling: The Use of Knowledge Representation Techniques for Requirements Specification, Ph. D. thesis, Dept. of Computer Science, Univ. of Toronto (1984)
22. Gross, D., Yu, E.: Evolving System Architecture to Meet Changing Business Goals: an Agent and Goal-Oriented Approach. ICSE-2001Workshop: From Software Requirements to Architectures (STRAW), Toronto, Canada. (2001) 13-21
23. Inverardi, P. et al.: The Teleservices and Remote Medical Care System (TRMCS): Case Study for the Tenth International Workshop on Software Specification and Design (IWSSD-10) (2000) http://www.ics.uci.edu/iwssd/case-study.pdf
24. Jarke, M., Kurki-Suonio, R.: Guest Editorial - Special Issue on Scenario Management. IEEE Transactions on Software Engineering, 24(12): (1998) 1033 -1035
25. Jarke, M.: Requirements Tracing - Introduction. Communications of the ACM, 41(12): (1998) 32-36
26. Jennings, N.R., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems, 1 (1998) 7-38
27. Kendall, E.A.: Agent Software Engineering with Role Modelling. In: Ciancarini, P., Wooldridge, M.J. (eds): Agent-Oriented Software Engineering: AOSE 2000. Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag. (2001) 163-170
28. Kethers, S.: Multi-Perspective Modeling and Analysis of Cooperation Processes. Ph.D. thesis. Technical University of Aachen (RWTH), Germany. (2001)
29. Medina-Mora, R., Winograd, T., Flores, R., Flores, F.: The Action Workflow Approach to Workflow Management Technology. Proc. Computer-Supported Cooperative Work. ACM Press. (1992) 281-288
30. Miksch, S., Cheng, K., Hayes-Roth, B.: An Intelligent Assistant For Patient Health Care, Proc. of the First Int. Conf. on Autonomous Agents (Agents'97) ACM Press (1997) 458-465
31. Mylopoulos, J., Borgida, A., Jarke, M., Kourbarakis, M.: Telos: A Language for Representing Knowledge About Information Systems. ACM Trans. on Information Systems 8(4) (1990) 325-362
32. Mylopoulos, J., Castro, J.: Tropos: A Framework for Requirements-Driven Software Development In J. Brinkkemper, A. Solvberg (eds.), Information Systems Engineering: State of the Art and Research Themes, Lecture Notes in Computer Science, Springer-Verlag (2000) 261-273
33. Mylopoulos, J.: Information Modeling in the Time of the Revolution. Information Systems 23(3-4): (1998) 127-155
34. Nuseibeh, B.A., Easterbrook, S. M.: Requirements Engineering: A Roadmap. In: Finkelstein, A.C.W. (ed): The Future of Software Engineering. (Companion volume to the proceedings of the 22nd Int. Conf. on Software Engineering, ICSE'00. IEEE Computer Society Press. (2000)
35. Omicini A.: SODA: Societies And Infrastructures In The Analysis And Design of Agent-based Systems. In: Ciancarini, P., Wooldridge, M.J. (eds): Agent-Oriented Software Engineering: AOSE 2000. Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag. (2001) 185-194
36. Parunak, H.V.D., Odell, J.: Representing social structures in UML. In this volume.
37. Petit, M.: A Multi-Formalism and Component-Based Approach to the Formal Modeling of Manufacturing Systems Requirements. Ph.D. thesis. University of Namur, Belgium. (2000)
38. Potts, C., Takahashi, K., Anton, A.: Inquiry-Based Requirements Analysis. IEEE Software, March (1994) 21-32

39. Robinson, W.N., & Volkov, S. Supporting the Negotiation Life-Cycle. Communications of the ACM, 41(5): (1998) 95-102
40. Ross, D.T.: Structured Analysis (SA): A Language for Communicating Ideas. IEEE Transactions on Software Engineering, SE-3(1) (1977) 16-34
41. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, Addison-Wesley (1998)
42. Simon, H.A.: The Sciences of the Artificial. MIT Press (1969).
43. Standish Group: Software Chaos (1995) http://www.standishgroup.com/chaos.html
44. Suchman, L.: Plans and Situated Actions: The Problem of Human-Machine Communication. Cambridge University Press (1987)
45. Szolovits, P., Doyle, J., Long, W.J., Kohane. I., Pauker, S.G.: Guardian Angel: Patient-Centered Health Information Systems. Technical Report MIT/LCS/TR-604 (1994)
46. Taveter, K.: From Descriptive to Prescriptive Models of Agent-Oriented Information Systems. 3rd Int. Workshop on Agent-Oriented Information Systems. Interlaken, Switzerland. (2001)
47. van Lamsweerde, A.: Requirements Engineering in the Year 2000: A Research Perspective. Proc. 22nd Int. Conf. on Software Engineering, June 2000, Limerick, Ireland (2000) 5-19
48. Wagner, G., Lespérance, Y., Yu, E., (eds): Agent-Oriented Information Systems 2000: Proceedings of the 2nd International Workshop. Stockholm, June 2000. iCue Publishing, Berlin (2000)
49. Wang, X., Lespérance, Y.: Agent-Oriented Requirements Engineering Using ConGolog and *i**. 3rd Int. Workshop on Agent-Oriented Information Systems. Montreal, Canada. (2001)
50. Weiss, G. (ed.): Multiagent Systems. MIT Press (1999)
51. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3 (3): (2000) 285-312
52. Yu, E.: Modelling Strategic Relationships for Business Process Reengineering. Ph.D. thesis. Dept. of Computer Science, University of Toronto. (1995)
53. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. Proc. of the 3rd IEEE Int. Symp. on Requirements Engineering (1997) 226-235.
54. Yu, E.: Why Agent-Oriented Requirements Engineering. In: Proc. of the 3rd Int. Workshop on Requirements Engineering: Foundations for Software Quality. Barcelona, Catalonia. E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur (1997)
55. Yu, E.: Agent Orientation as a Modelling Paradigm. Wirtschaftsinformatik 43(2) (2001) 123-132.
56. Yu, E., Mylopoulos, J.: Understanding "Why" in Software Process Modelling, Analysis, and Design, Proc. 16th Int. Conf. Software Engineering, Sorrento, Italy, (1994) 159-168
57. Yu, E., Mylopoulos, J.: Why Goal-Oriented Requirements Engineering, Proc. of the 4th Int. Workshop on Requirements Engineering: Foundations of Software Quality, Pisa, Italy. E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur (1998) 15-22
58. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Organisational Abstractions for the Analysis and Design of Multi-Agent Systems. In: Ciancarini, P., Wooldridge, M.J. (eds): Agent-Oriented Software Engineering: AOSE 2000. Lecture Notes in Computer Science, Vol. 1957. Springer-Verlag. (2001) 235-251