From Object-Oriented to Goal-Oriented

Requirements Analysis

Goal-oriented and object-oriented analysis should be seen as complementary, the former focusing on the early stages of requirements analysis … the latter on late stages.

The growing influence of object-oriented programming on programming practice has led to the rise of a new paradigm for system and software requirements analysis, popularly known as object-oriented analysis (OOA). This paradigm adopts ideas from object-oriented programming and blends them with ideas from semantic data modeling and knowledge representation (notably semantic networks) into a modeling framework that is more powerful than traditional techniques such as data flow diagrams, structured analysis, and the like.

The first object-oriented analysis techniques were proposed more than 10 years ago. The Object-Oriented Systems Analysis (OOSA) technique [12] adopts the Entity-Relationship (ER) model to capture the declarative aspects of a software system. This was soon followed by two new proposals, Object-Oriented Analysis [3] and the Object-Oriented Modeling Technique (OMT) [11], which support the modeling of declarative, behavioral as well as interactive aspects of a software system. Today, there are dozens of like-minded techniques and commercial tools founded on the OO way of thinking that support development from requirements analysis to implementation. Indeed, the great promise of OOA is that the whole software development process can be streamlined and simplified by having the same building blocks (objects, classes, methods, messages, inheritance and the like) used in all phases of development, from requirements to implementation. A recent proposal, the Unified Modeling Language (UML)—see www.rational.com/uml—attempts to integrate features of the more preeminent models in OOA, thereby enhancing reusability and consolidating the growing OOA market.

Why is OOA popular? In a nutshell, because it significantly advances the state of practice in requirements modeling. The prac-

John Mylopoulos, Lawrence Chung, and Eric Yu

tice of systems analysis was characterized 10 years ago by a mixed bag of isolated modeling techniques (data flow diagrams, ER diagrams, state transition diagrams) that were used to capture the rich information that needs to be modeled, analyzed and understood before a software system is actually built. These techniques generally offered little help for structuring requirements models, to ensure that they were readily understandable, extensible and amenable to analysis. In contrast to this situation, OOA techniques offer a coherent framework which integrates a comprehensive set of modeling concepts for capturing declarative, behavioral, and interactive aspects of a system.[1] In addition, OOA techniques strongly support two structuring mechanisms, generalization and aggregation, in terms of which a modeler can organize and manage the immense amount of information captured by her models. A final important reason for the popularity of OOA techniques rests with the popularity of OO programming itself. Earlier requirements analysis techniques were inspired by, and founded on, structured programming concepts. In a programming world that is increasingly turning to object orientation, such techniques seemed out of date and had to be replaced.

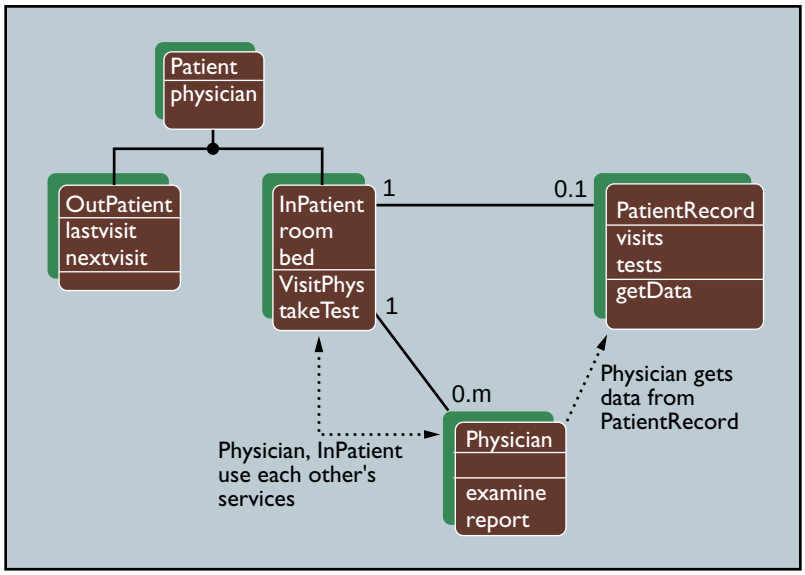

Figure 1. A small portion of a hospital model for requirements analysis

Since OOA techniques are intended for requirements analysis,[2] the models built in terms of these techniques comprise models of a real-world environment within which the new system will eventually operate, that is, an environment consisting of people, work processes, material things, software systems and the like. For example, Figure 1 models aspects of a hospital setting, such as patients and physicians. In the figure, a rounded rectangle refers to a class of objects, whose name, attributes and services respectively appear in the upper, middle and lower part of the rectangle; a semi-ellipse specifies one or more specializations of a class, a solid arc denotes a relationship between classes; and a broken arrow indicates that one class uses a service from another. According to the figure, the class `Patient` has one attribute, `physician`, and no associated services, as well as two specializations, `OutPatient` and `InPatient`. Moreover, `InPatient` has two additional attributes, `room` and `bed`, two services, `visitPhys(ician)` and `takeTest`, and is related to `PatientRecord` through a one-to-one relationship. During requirements analysis the use of such diagrams, along with natural language, ensures that different stakeholders (patients, hospital management, requirements analysts, hospital staff, and so forth) agree on the relevant objects and relationships (and other things, such as hospital procedures and policies.) During system design, some of the classes in the requirements model may need to be projected into a design. For example, if the new system needs to keep track of information about physicians, then the design may include a class `PhysicianInfo` which will represent information about physicians through its instances. Note that the `PhysicianInfo` class, which describes a component of the system design, may not have the same attributes or services as the `Physician` class, which models one aspect of a hospital setting in the real world.

Any modeling technique "colors" the view of its users because it offers only a limited number of primitive concepts for modeling its intended subject matter. The kinds of information that can be captured by the analyst are then characterized by precisely these concepts. The purpose of this article is to offer a sketch of the concept of the *softgoal*, for modeling and analyzing non-functional requirements, and to show how this can contribute toward a foun-

[1] See [9] for a discussion of how OOA modeling features improve on their structured analysis predecessors.

[2] Actually, most OOA techniques address both requirements analysis and design, often with little to say about the boundary between these two important software development phases; for our discussion, we focus exclusively on requirements analysis.

dation for a *goal-oriented* requirements analysis, which complements and enriches OOA.

Non-functional requirements (or *quality attributes, qualities,* or more colloquially *"-ilities"*) are global qualities of a software system, such as flexibility, maintainability, usability, and so forth. Such requirements are usually stated only informally, are often controversial (for example, management wants a secure system but staff desires user-friendliness), are difficult to enforce during design and implementation, and are diffcult to validate. Not surprisingly, unmet quality requirements constitute an important failure factor for software development projects.

Modeling the World

Since computer applications must ultimately be useful in the real world, modeling a part of that world, the *application domain*, has been a major preoccupation in several areas of computer science, such as data modeling in databases, knowledge representation in artificial intelligence (AI) and requirements modeling in software engineering. Over the years, hundreds of notations, often referred to as *conceptual* or *semantic* models, have been proposed for such modeling tasks. In general, a *conceptual model* comprises a collection of:

- *Primitive terms*, which specify a set of basic building blocks for constructing symbol structures;
- *Structuring mechanisms* for assembling and organizing symbol structures;
- *Primitive operations*, for constructing and querying symbol structures;
- General *integrity rules*, which define the set of consistent symbol structure states, or changes of states. These are accompanied by interpretation rules and usage guidelines.

For example, Peter Chen's original ER model offers `Entity`, `Relationship` and `Attribute` as primitive terms, supports a limited form of classification (because entities and relationships are instances of entity and relationship types), offers primitive operations for creating new entity or relationship types or instances thereof, and supports cardinality constraints for relationships, such as "every child has up to two parents." An important extension of the model, the Extended Entity-Relationship model, supports all the features of the ER model and also offers generalization and aggregation for structuring purposes. The ER model was proposed at the first Very Large Databases conference in 1975. The ER model is one of the first *semantic* data models because it assumes that the domain to be modeled consists of entities and relationships, unlike the relational model, which makes no assumptions at all about the domain.

In the field of AI, *semantic networks* were proposed almost 10 years earlier by Ross Quillian's Ph.D. dissertation (completed in 1966), as suitable symbolic models of human memory. Semantic networks are directed, labeled graphs whose nodes represent concepts while links represent binary relationships. Quillian's proposal actually supported generalization as a structuring mechanism, and also provided for inheritance of attributes. Semantic networks were upgraded with procedural attachments and other facilities to form frame-based knowledge representation languages. Along a different path, they were combined with a logical sublanguage for specifying formal properties of defined classes and/or tokens. Description logics, a popular form of knowledge representation language today, originated from this line of research.

In software engineering, Douglas Ross proposed the Structured Analysis and Design Technique (SADT™) in the mid-1970s as a "language for communicating ideas" [10]. According to SADT, the world consists of activities and data, which are both represented by boxes and arrows. Each activity consumes some data, represented through input arrows entering from the left, and produces some data, represented through output arrows exiting from the right. In addition, each activity has associated data that controls its execution, but is neither consumed nor produced, and some external agent (hardware or human) that executes it. Analogous diagrams can be used to model data in a dual fashion. Other structured analysis techniques, such as the popular data flow diagrams, adopted ideas from SADT but focused more specifically on the modeling of information flows within an organization, instead of SADT's all-inclusive modeling framework. A thorough review of the history and features of conceptual models can be found in [9].

Requirements engineering was born in the mid-1970s, partly thanks to Ross and his SADT proposal, and partly thanks to others who established through empirical study that "the rumored 'requirements problems' are a reality." The case for world modeling during requirements analysis was eloquently articulated [6], that software development methodology starts with a "model of reality with which [the system] is concerned." Sol Greenspan's RML (Requirements Modeling Language) [5] formalizes SADT by using ideas from semantic networks and semantic data models. The result is a requirements modeling language in which entity and

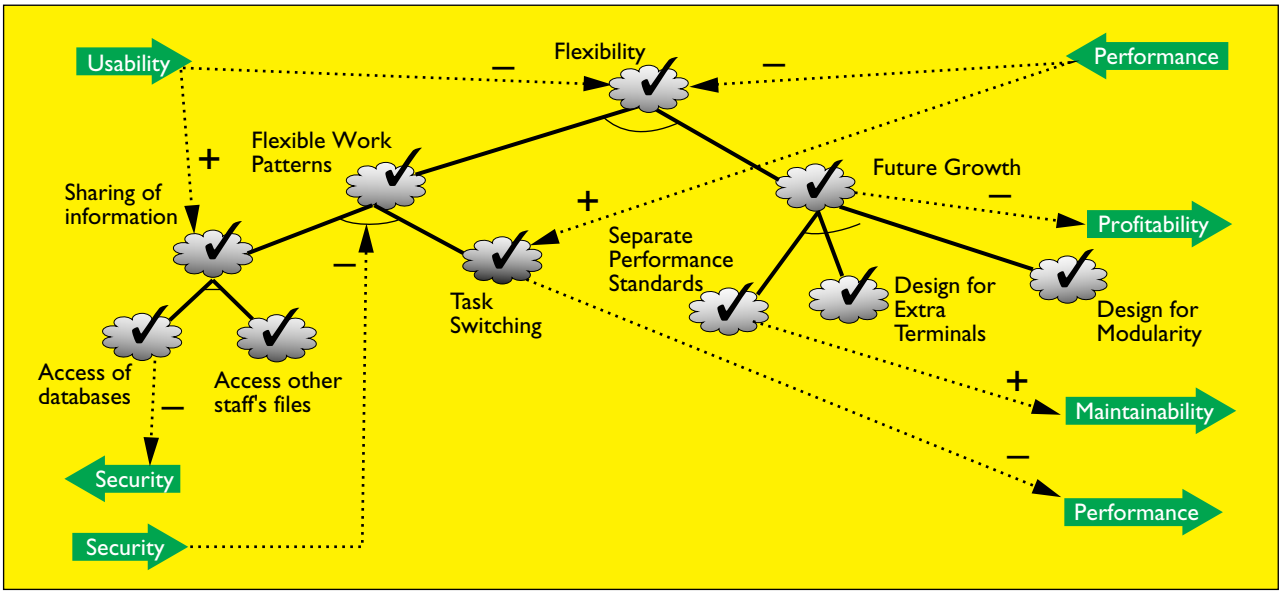

Figure 2. The (partial) result of nonfunctional requirements analysis for an office support system. (This figure was adopted from coursework by L. Gibbons and J. Spiess, prepared for a graduate-level course taught by Eric Yu.)

activity classes are organized into generalization hierarchies, and which in a number of ways predates OOA techniques by several years. More recently, there have been many proposals for goal-oriented approaches to requirements engineering, including [7] which is semiformal and pragmatic, also KAOS [4], which is more long-term and heavyweight. KAOS provides facilities for describing a variety of concepts such as goals, agents, alternatives, events, actions, existence modalities and agent responsibilities. Moreover, KAOS relies heavily on a metamodel to provide a self-descriptive and extensible modeling framework. Both RML and KAOS exploit many of the same modeling constructs used by OOA notations, though neither was conceived within the OOA community. However, unlike OOA techniques, RML and KAOS are formal requirements modeling languages. Their formal semantics constitute a solid foundation for building sophisticated analysis tools.

This is a very sketchy account of a very active research area. For more details readers are directed to the proceedings of the IEEE International Symposia and Conferences on Requirements Engineering launched in 1993, also to the journal *Requirements Engineering* published by Springer-Verlag since 1996. An early and influential collection of papers on the topic of conceptual modeling can be found in [1].

AND $(G,\{G_1,G_2,\ldots,G_n\})$	-- goal **G** is satisficed when all of $G_1,G_2,\ldots,G_n$ are satisficed and there is no negative evidence against it; -- goal **G** is unsatisficed and there is one of $G_1,G_2,\ldots,G_n$ is unsatisficed and there is no positive evidence for it.
OR $(G,\{G_1,G_2,\ldots,G_n\})$	-- goal **G** is satisficed when one of $G_1,G_2,\ldots,G_n$ is satisficed and there is no negative evidence against it; -- goal **G** is unsatisficeable if all of $G_1,G_2,\ldots,G_n$ are unsatisfieceable and there is no positive evidence for it.
$+(G_1, G_2)$	-- goal G_1 contributes positively to the satisficing of goal G_2.
$-(G_1, G_2)$	-- goal G_1 contributes negatively to the satisficing of goal G_2.

Table 1. Softgoal relationships

Satisficing Softgoals

Imagine that you have been asked by your client to conduct a requirements analysis for a new system[3] intended to support various office functions within its organization, including scheduling meetings. Right from the start, the client is very clear that any new system should be highly usable, flexible and adaptable to the work patterns of individual users and that its introduction should create as little disruption as possible. You understand that your task calls for modeling the objects and activities in the operational environment of the new system, including people, office procedures, information items being created or used and the like. You also know that other stakeholders in the project need to be consulted, such as the office staff for whom the new

[3] That is the hardware and software (to be built) that will support meeting scheduling.

system is intended. But how are you going to deal with the client's objectives of having a usable and flexible system? You realize that these objectives are all-important, but unfortunately get little guidance from your favorite OOA technique on what to model and how to include these objectives in your analysis.

To bring flexibility and usability into the requirements analysis process, we first need some way to represent them, along with their respective interrelationships. For purposes of illustration, we adopt the *Non-Functional Requirements* (NFR) framework, which centers around the notion of softgoal [8].

The concept of *goal* is used extensively in AI where a goal is satisfied absolutely when its subgoals are satisfied, and that satisfaction can be automatically established by an algorithm. To support the relative, ill-defined, tentative and contradictory nature of non-functional requirements, however, we need a looser notion of goal. Softgoals are goals that do not have a clear-cut criterion for their satisfaction. We will say that softgoals are *satisficed*[4] when there is sufficient positive and little negative evidence for this claim, and that they are *unsatisficeable* when there is sufficient negative evidence and little positive support for their satisficeability. Sometimes the evidence is sufficiently strong for the decision for softgoal satisficeability to be made automatically without human intervention. In other cases, when there is weak or conflicting evidence, the decision may have to be made interactively by the stakeholders in the requirements analysis process.

In analyzing non-functional requirements, one does not analyze softgoals independently of one another, but rather in relation to each other. Two obvious types of relationships are the `AND` and `OR` goal relationships comparable to the ones traditionally used in AI planning. There can also be other, looser relationships in which one softgoal subsumes, prevents, or contributes to the fulfillment of another. For this discussion, we will only use the four relationships shown in Table 1. With these preliminaries, we are ready to describe one of the constituents of goal-oriented requirements analysis.

Non-functional requirements analysis. This form of analysis begins with softgoals that represent non-functional requirements agreed upon by the stakeholders, say `Usability`, `Flexibility`, etc. Then one refines these by using decomposition methods. These methods may be generic, derived from general expertise about flexibility, security, and the like. They may also be domain-specific (specific to meeting scheduling), or even project-specific (decided upon jointly by the stakeholders of a project). Let's consider `Flexibility` (of the new system) for illustration purposes. This softgoal might be decomposed to two other softgoals: the first, `FlexibleWorkPatterns[staff]`, calls for flexibility in the work patterns allowed by the new system for all staff, while the second, `FutureGrowth`, calls for a system architecture that can accommodate future growth. Along similar lines, the `Flexible-`

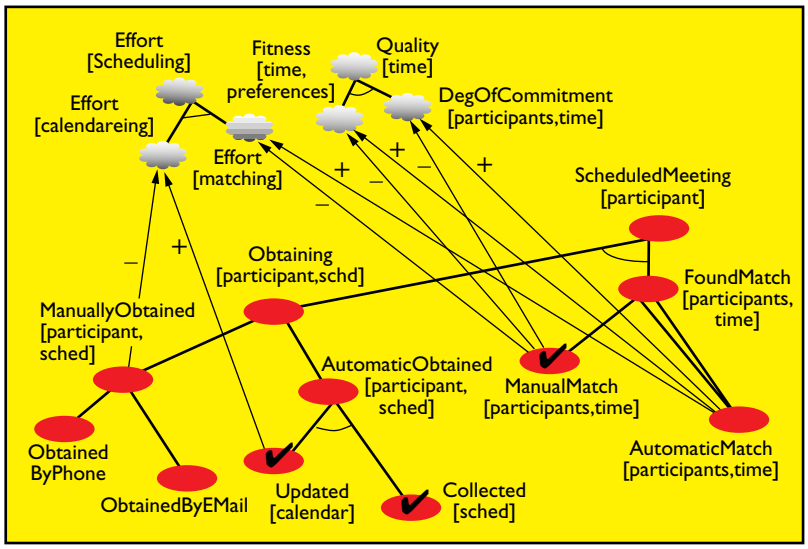

Figure 3. Functional requirements analysis for the office support system

`WorkPatterns` softgoal is further decomposed to `SharingOfInformation`, and `TaskSwitching`, among staff. Using such an analysis, the softgoal tree structure for `Flexibility` is created, as shown in Figure 2.

However, `Flexibility` is not the only non-functional requirement desired by the client. Goal trees for usability, performance, security, etc. need to be likewise elaborated. As one refines these, there is bound to be interference among softgoals belonging to different softgoal tree structures. Accordingly, another phase of the non-functional requirements analysis involves finding lateral relationships between the softgoals of individual softgoal trees. For instance, performance goals generally interfere with flexibility ones. Moreover, allowing general

[4]See H.A. Simon's *The Sciences of the Artificial, 2d edition,* published by MIT Press (1981).

access to databases interferes with security goals at some level. Everyone in turn realizes that security softgoals generally interfere with flexible work patterns. The end result of this analysis is that lateral relationships are created among the softgoals of different softgoal trees, marking positive as well as negative interferences. The collection of softgoal trees has now been turned into a softgoal graph structure, with possible cyclic paths.

The final step of this analysis is to pick particular leaf nodes of each softgoal tree structure so that all root softgoals are satisficed. For instance, in Figure 2, checked leaf softgoals are all picked to be accommodated by the new system. Without interferences, the algorithm for determining whether the root of an AND/OR goal tree has been satisficed, given that some of its leaf nodes are, is straightforward. With interferences, and the presence of multiple types of relationships representing different forms of positive or negative interference, the algorithm for determining the label of each root node uses a form of label propagation, adopted from qualitative reasoning techniques in AI [8]. The selection of a set of leaf nodes represents a set of design decisions imposed on the new system.

ure shows a possible refinement of the Scheduled-Meeting goal. The goal has been refined to two (AND) subgoals, Obtained[schedule] and FoundMatch. These are further refined, depending on whether these tasks are to be done manually or by the new system. But how are we to choose among alternative designs for the office procedure that handles meeting schedulings? Once again, qualities play a role in the selection process, as shown with the two quality softgoal trees of the upper end of the figure, and the positive and negative influences from different design alternatives. Once we have settled on a particular set of leaf goals, as shown in Figure 3 with the check-marked functional goals, we have defined tasks to be carried out by the new system, and possibly by office workers as well. This figure also shows that the internal structure of the softgoals can be further analyzed, for example, by separating a quality sort (Flexibility) from the object it is applied to (System), and from other attributes. This allows relevant knowledge to be brought to bear on the analysis process: from very generic ("To achieve quality X for a system, try to achieve X for all its components") to very specific ("To achieve effectiveness of a software review meeting, all stakeholders must be present"). Knowledge structuring mechanisms such as classification, generalization, or aggregation, for example, can be used to organize the available know-how for supporting such a goal-oriented analysis process. A more detailed example on the use of softgoals in facilitating software evolution can be found in [2].

> { Goal-oriented analysis focuses on the description and evaluation of alternatives and their relationship to the organizational objectives. }

Functional requirements analysis. Traditionally, requirements have been classified as functional or non-functional. Functional requirements are also goals. For instance, functional requirements for the office support system might include: "System must support meeting scheduling," or "System will generate reimbursements for travel."

Such requirements will lead to particular functions for the new system, such as maintaining a database of schedules for all office staff, or finding a suitable meeting time given the scheduling constraints of all participants. In Figure 3, functional goals are represented as ellipses to distinguish them from their non-functional, cloud-like cousins. Given an initial set of functional goals, one would look for ways to satisfy them through a process to be carried out by the new system and/or workers within the office and/or other, existing systems.

The goal tree structure in the lower half of the fig-

Conflict analysis. As indicated previously, softgoals are bound to conflict with each other. For instance, the softgoals of Figure 2 labeled Access-AllDatabases and AccessOtherStaffsFiles contribute to satisficing Flexibility, but interfere with security goals. A conflict can also involve functional goals: making timetables publicly available, for example, to facilitate the scheduling of meetings could interfere with security and/or privacy softgoals. We are only beginning to understand the importance and depth of the conflict analysis problem.

Goal-oriented analysis amounts to an intertwined execution of the three types of analysis sketched here, namely analyses of non-functional requirements as softgoals, of functional requirements as goals, and conflict analysis. The analysis can be declared complete when all relevant goals (soft or otherwise) have been operationalized in terms of constraints on, and functions to be performed by, the new system.

Conclusion

We have placed OOA techniques within the context of other notations and methods intended to model aspects of the real world. On that basis, we have argued that adoption of an alternative set of primitive modeling concepts, such as those of softgoal and goal, can lead to a rather different kind of analysis than those advocated by OOA techniques. Moreover, this kind of analysis is very important because it deals with non-functional requirements and relates them to functional ones. As readers may have already concluded, goal-oriented analysis focuses on the description and evaluation of *alternatives* and their relationship to the organizational objectives behind a software development project. As many within the requirements engineering research community have argued, capturing these interdependencies between organizational objectives and the detailed software requirements can facilitate the tracing of the origins of requirements, and can help make the requirements process more thorough, complete, and consistent. Preliminary empirical studies suggest that goal-oriented analysis can indeed lead to a more complete requirements definition than OOA techniques can. Moreover, our own experiences in analyzing the requirements and architectural design for a large (telecommunications) software system confirm that goal-oriented analysis can greatly facilitate and rationalize early phases of the software design process.

Of course, OOA techniques still have a place within a requirements analysis process even if one adopts goal-oriented analysis. After all, OOA models define the objects and activities mentioned in the detailed requirements for the new system. So goal-oriented analysis and OOA should be seen as complementary, the former focusing on the early stages of requirements analysis and on the rationalization of the development process, the latter on late stages of requirements analysis. The KAOS methodology gives an excellent sample of how the two types of analysis coexist and complement each other.

Traditionally, requirements analysis practice has been driven by the programming paradigm of the day. Thus, in the days of structured programming, structured analysis ruled, whereas today interest is shifting to OOA. Given the importance of requirements analysis to the success of any large software development project, perhaps it is time to turn things around: suppose we let the concepts and techniques of goal-oriented analysis drive the design and implementation techniques that follow. What would such a software development methodology look like? Perhaps it will be based on software architectures that share some of the characteristics of human organizations and be grounded in the concepts of agent, goal, and of course softgoal. Actually, agent programming *is* gaining in popularity as the programming paradigm for network computing, so the possibility of a new development methodology grounded on goal-oriented analysis and agent-based design and implementation may not be as far-fetched as it might seem. C

References

1. Brodie, M., Mylopoulos, J., and Schmidt, J., Eds. *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages.* Springer Verlag, 1984.
2. Chung, L., Nixon, B., Yu, E. Dealing with change: An approach using non-functional requirements. *Requirements Engineering 1*, 4 (1996), 238–260.
3. Coad, P. and Yourdon, E. *Object-Oriented Analysis.* Yourdon Press, Englewood Cliffs, NJ, 1990.
4. Dardenne, A., van Lamsweerde, A., and Fickas, S. Goal-directed requirements acquisition. *Science of Computer Programming 20* (1993), 3–50.
5. Greenspan, S., Borgida, A., and Mylopoulos, J. A requirements modelling language and its logic. *Information Systems 11*, 1 (1986), 9–23.
6. Jackson, M.A. *System Development.* Prentice Hall, London, 1983.
7. Kaindl, H. A practical approach to combining requirements definition and object-oriented analysis. *Annals of Software Engineering 3* (1997), 319–343.
8. Mylopoulos, J., Chung, L. and Nixon, B. Representing and using non-functional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.* (June 1992).
9. Mylopoulos, J. Information modeling in the time of the revolution. *Info. Syst. 23,* 3–4 (June 1998), 127–156.
10. Ross, D. Structured analysis: A language for communicating ideas. *IEEE Trans. Softw. Eng. 3*, 1 (Jan. 1977).
11. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
12. Shlaer, S. and Mellor, S. *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice Hall, 1988.

John Mylopoulos (jm@cs.toronto.edu) is professor of computer science at the University of Toronto.
Lawrence Chung (chung@utdallas.edu) is an assistant professor in the department of computer science at the University of Texas at Dallas.
Eric Yu (yu@fis.utoronto.ca) is an assistant professor in the Faculty of Information Studies at the University of Toronto.