# Goal-Oriented Requirements Engineering: An Overview of the Current Research

by
Alexei Lapouchnian

Department of Computer Science
University Of Toronto
28.06.2005

# 1. Introduction and Background

## 1.1 Requirements Engineering

The main measure of the success of a software system is the degree to which it meets its purpose. Therefore, identifying this purpose must be one of the main activities in the development of software systems. It has been long recognized that inadequate, incomplete, ambiguous, or inconsistent requirements have a significant impact on the quality of software. Thus, Requirements Engineering (RE), a branch of software engineering that deals with elicitation, refinement, analysis, etc. of software systems requirements gained a lot of attention in the academia as well as in the industry.

One of the oldest definitions of Requirements Engineering says that *"requirements definition is a careful assessment of the need that a system is to fulfill. It must say why a system is needed, based on current or foreseen conditions, which may be internal operations or external market. It must say what system features will serve and satisfy this context. And it must say how the system is to be constructed"* [55]. Thus, requirements engineering must address the reasons why a software system is needed, the functionalities it must have to achieve its purpose and the constraints on how the software must be designed and implemented. Today we can find many definitions of requirements engineering. For example, requirements engineering is defined in [47] as the process of discovering the purpose of software systems by identifying stakeholders ("people or organizations who will be affected by the system and who have a direct or indirect influence on the system requirements" [43]) and their needs and by documenting these in a form that is amenable to analysis, communication, and subsequent implementation. Requirements engineering is defined in [65] as "the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and their evolution over time and across software families".

van Lamsweerde [26] describes the following intertwined activities that are covered by requirements engineering (a similar set of activities is also featured in [47]):

- *Domain analysis*: the environment for the system-to-be is studied. The relevant stakeholders are identified and interviewed. Problems with the current system are discovered and opportunities for improvement are investigated. Objectives for the target system are identified.
- *Elicitation*: alternative models for the target system are analyzed to meet the identified objectives. Requirements and assumptions on components of such models are identified. Scenarios could be involved to help in the elicitation process.
- *Negotiation and agreement*: alternative requirements and assumptions are evaluated; risks are analyzed by the stakeholders; the best alternatives are selected.
- *Specification*: requirements and assumptions are formulated precisely.
- *Specification analysis*: the specifications are checked for problems such as incompleteness, inconsistency, etc. and for feasibility.
- *Documentation*: various decisions made during the requirements engineering process are documented together with the underlying rationale and assumptions.

- *Evolution*: requirements are modified to accommodate corrections, environmental changes, or new objectives.

Modeling appears to be a core process in RE. The existing system/organization as well as the possible alternative configurations for the system-to-be are typically modeled. These models serve as a basic common interface to the various activities above [26]. Requirements modeling is the process of "building abstract descriptions of the requirements that are amenable to interpretation" [47]. Modeling facilitates in requirements elicitation by guiding it and helping the requirements engineer look at the domain systematically. Domain models help communicate requirements to customers, developers, etc. The models also allow for requirements reuse within the domain. The presence of inconsistencies in the models is indicative of conflicting and/or infeasible requirements. Models also provide a basis for requirements documentation and evolution. While informal models are analyzed by humans, formal models of system requirements allow for precise analysis by both the software tools and humans.

One of the main difficulties of requirements engineering is the imprecise, informal nature of requirements and the fact that it "concerns translation from informal observations of the real world to mathematical specification languages" [62]. Requirements for a system exist in a certain social context and therefore during requirements elicitation a high degree of communication, negotiation, and other interaction skills is needed. For example, various stakeholders have different, possibly (and usually) conflicting points of view on what the system-to-be should deliver. Thus, getting stakeholders to agree on requirements is an important step of the process and requires a good social and communication skills. Requirements engineers should also be very careful while gathering the requirements: the same words and phrases used by different stakeholders may mean different things for them. One way to overcome this problem is to construct a common ontology to be used by all the stakeholders. Another way is to model the environment formally. One approach for carefully describing the environment using ground terms, designations and definitions is presented in [24]. A brief overview of the requirements elicitation techniques including traditional, model-driven, etc. is presented in [47].

## 1.2 The Nature of Requirements

Despite the fact that the idea of requirements had been around for quite some time (e.g., [55]), a number of important properties of software requirements remained unknown until mid-1990s. In [24], Michael Jackson makes the distinction between the *machine*, which is one or more computers that behave in a way to satisfy the requirements with the help of software, and the *environment*, which is the part of the world with which the machine will interact and in which the effects of the machine will be observed. When the machine is put into its environment, it can influence that environment and be influenced by that environment only because they have some *shared phenomena* in common. That is, there are events that are events of both and there are states that are states of both. Requirements are located in the environment. They are conditions over the events and states of the environment (or, possibly, the events and states in the shared phenomena) and can be formulated in a language accessible to stakeholders. So, the requirements can be stated without referring to the machine. In order to satisfy its requirements the machine acts on the shared phenomena.

Through the properties of the environment (causal chains) it can also indirectly affect the private phenomena of the environment through which the requirements are expressed. Therefore, the description of the requirements must describe the *desired* (optative) conditions over the environment phenomena (*requirements*) and also the *given* (indicative) properties of the environment (*domain properties*) that allow the machine participating only in the shared phenomena to ensure that the requirements are satisfied in the environment phenomena [24] (Parnas [48] independently made the same distinction between requirements and domain properties). Another distinction made by Jackson and Parnas was between the requirements and software *specifications*, which are formulated in terms of the machine phenomena in the language accessible to software developers. Further, important distinction must be made about requirements and environment *assumptions* (sometimes called *expectations*). Even though they are both optative, requirements are to be enforced by the software, while assumptions are to be enforced by agents in the environment. The assumptions specify what the system expects of its environment. For example, one could have an assumption about the behaviour of a human operator of the system.

Requirements engineering is generally viewed as a process containing two phases. The *early requirements phase* concentrates on the analysis and modeling of the environment for the system-to-be, the organizational context, the stakeholders, their objectives and their relationships. A good analysis of the domain is extremely important for the success of the system. Understanding the needs and motivations of stakeholders and analyzing their complex social relationships helps in coming up with the correct requirements for the system-to-be. Domain models are a great way to organize the knowledge acquired during the domain analysis. Such models are reference points for the system requirements and can be used to support the evolution of requirements that stems from changes in the organizational context of the system. The *late requirements phase* is concerned with modeling the system together with its environment. The system is embedded into the organization; the boundaries of the system and its environment are identified and adjusted, if needed; the system requirements and assumptions about the environment are identified. The boundary between the system and its environment is initially not well defined. The analyst will try to determine the best configuration of the system and its environment to reliably achieve the goals of the stakeholders. Putting too much functionality into the system could make it, for example, too complex and hard to maintain and evolve, while making too many assumptions about the environment may be unrealistic.

Within the realm of requirements there exists another important division, *functional* versus *non-functional* requirements. Functional requirements specify the functions or services of the system. On the other hand, non-functional (quality) requirements (NFRs) represent software system qualities (e.g., security, ease of use, maintainability, etc.) or properties of the system as a whole. NFRs are generally more difficult to express in an objective and measurable way. Thus, their analysis is also more difficult.

# 2. Overview of Goal-Oriented Requirements Engineering

## 2.1 From Traditional to Goal-Oriented RE

In the recent years, the popularity of goal-oriented requirements engineering approaches has increased dramatically. The main reason for this is the inadequacy of the traditional systems analysis approaches (e.g., [54][15][56]) when dealing with more and more complex software systems. At the requirements level, these approaches treat requirements as consisting only of processes and data and do not capture the rationale for the software systems, thus making it difficult to understand requirements with respect to some high-level concerns in the problem domain. Most techniques focus on modeling and specification of the software alone. Therefore, they lack support for reasoning about the *composite system* comprised of the system-to-be and its environment. However, incorrect assumptions about the environment of a software system is known be responsible for many errors in requirements specifications [34]. Non-functional requirements are also in general left outside of requirements specifications. Additionally, traditional modeling and analysis techniques do not allow alternative system configurations where more or less functionality is automated or different assignments of responsibility are explored, etc. to be represented and compared. Goal-Oriented Requirements Engineering (GORE) attempts to solve these and other important problems.

It is important to note that goal-oriented requirements elaboration process ends where most traditional specification techniques would start [34]. Overall, GORE focuses on the activities that precede the formulation of software system requirements. The following main activities are normally present in GORE approaches: goal elicitation, goal refinement and various types of goal analysis, and the assignment of responsibility for goals to agents. We will talk about these and other activities later in the document.

## 2.2 The Main Concepts

Most of early RE research concentrated on what the software system should do and how it should do it. This amounted to producing and refining fairly low-level requirements on data, operations, etc. Even though the need to capture the rationale for the system under development was evident from the early definitions of requirements (e.g., “requirements definition must say why a system is needed” [55]), little attention was given in the RE literature to understanding why the system was needed and whether the requirements specification really captured the needs of the stakeholders. Not enough emphasis was put on understanding the organizational context for the new system. In general, the tendency in the RE modeling research has been to abstract the low-level programming constructs to the level of requirements rather than pushing requirements abstractions down to the design level [9].

This explains why the stakeholders with their needs and the rest of the social context for the system could not be adequately captured by requirements models.

*Goals* have long been used in artificial intelligence (e.g., [46]). Yet, Yue was likely the first one to show that goals modeled explicitly in requirements models provide a criterion for requirements completeness [64]. A model of human organizations that views humans and organizations as goal-seeking entities can be seen as the basis for goal modeling. This model is dominant in the information systems field [10]. However, it remains rather implicit in the RE literature.

There are a number of goal definitions in the current RE literature. For example, van Lamsweerde [26] defines a goal as an objective that the system should achieve through cooperation of agents in the software-to-be and in the environment. Anton [1] states that goals are high-level objectives of the business, organization or system; they capture the reasons why a system is needed and guide decisions at various levels within the enterprise.

An important aspect of requirements engineering is the analysis of non-functional (quality) requirements (NFRs) [45]. NFRs are usually represented in requirements engineering models by *softgoals*. There is no clear-cut satisfaction condition for a softgoal. Softgoals are related to the notion of *satisficing* [57]. Unlike regular goals, softgoals can seldom be said to be accomplished or satisfied. For softgoals one needs to find solutions that are "good enough", where softgoals are satisficed to a sufficient degree. High-level non-functional requirements are abundant in organizations and quite frequently the success of systems depends on the satisficing of their non-functional requirements.

Goal-oriented requirements engineering views the system-to-be and its environment as a collection of active components called *agents*. Active components may restrict their behaviour to ensure the constraints that they are assigned. These components are humans playing certain roles, devices, and software. As opposed to passive ones, active components have a choice of behaviour. In GORE, agents are assigned responsibility for achieving goals. A *requirement* is a goal whose achievement is the responsibility of a single software agent, while an *assumption* is a goal whose achievement is delegated to a single agent in the environment. Unlike requirements, expectations cannot be enforced by the software-to-be and will hopefully be satisfied thanks to organizational norms and regulations [27]. In fact, requirements "implement" goals much the same way as programs implement design specifications [34]. Agent-based reasoning is central to requirements engineering since the assignment of responsibilities for goals and constraints among agents in the software-to-be and in the environment is the main outcome of the RE process [26].

## 2.3 Benefits of Goal Modeling

There are a number of important benefits associated with explicit modeling, refinement, and analysis of goals (mostly adapted from [27]):

- It is important to note that GORE takes a wider system engineering perspective compared to the traditional RE methods: goals are prescriptive assertions that should hold in the system made of the software-to-be *and its environment*; domain properties and expectations about the environment are explicitly captured during the requirements elaboration process, in addition to the usual software requirements specifications. Also, goals provide rationale for requirements that operationalize them. Thus, one of the main benefits of goal-oriented requirements engineering is the added support for the early requirements analysis.

- Goals provide a precise criterion for *sufficient completeness* of a requirements specification. The specification is complete with respect to a set of goals if all the goals can be proven to be achieved from the specification and the properties known about the domain [64].

- Goals provide a precise criterion for requirements *pertinence*. A requirement is pertinent with respect to a set of goals in the domain if its specification is used in the proof of one goal at least [64]. Even without the use of formal analysis methods, one can easily see with the help of goal models whether a particular goal in fact contributes to some high-level stakeholder goal.

- A goal refinement tree provides *traceability* links from high-level strategic objectives to low-level technical requirements.

- Goal modeling provides a natural mechanism for structuring complex requirements documents [27].

- One of the concerns of RE is the management of conflicts among multiple *viewpoints* [16]. Goals can be used to provide the basis for the detection and management of conflicts among requirements [25][51].

- A single goal model can capture variability in the problem domain through the use of alternative goal refinements and alternative assignment of responsibilities. Quantitative and qualitative analysis of these alternatives is possible.

- Goal models provide an excellent way to communicate requirements to customers. Goal refinements offer the right level of abstraction to involve decision makers for validating choices being made among alternatives and for suggesting other alternatives.

- Separating stable from volatile information is also important in requirements engineering. A number of researches point out that goals are much more stable than lower-level concepts like requirements or operations [1][27]. A requirement represents one particular way of achieving some goal. Thus, the requirement is more likely to evolve towards a different way of achieving that same goal than the goal itself. In general, the higher level the goal is the more stable it is.

# 3. The Main GORE Approaches

In this section, we describe the core of the main GORE approaches while Section 4 provides an overview of some of the active areas in the goal-oriented requirements engineering research.

## 3.1 The NFR Framework

The NFR framework was proposed in [45] and further developed in [12]. The NFR framework (as it is evident from its name) concentrates on the modeling and analysis of non-functional requirements. The goal of the framework is to put non-functional requirements foremost in developer's mind [12]. The framework aims at dealing with the following main activities: capturing NFRs for the domain of interest, decomposing NFRs, identifying possible NFR operationalizations (design alternatives for meeting NFRs), dealing with ambiguities, tradeoffs, priorities, and interdependencies among NFRs, selecting operationalizations, supporting decisions with design rationale, and evaluating impact of decisions. The main idea of the approach is to systematically model and refine non-functional requirements and to expose positive and negative influences of different alternatives on these requirements.

The framework supports three types of softgoals. *NFR softgoals* represent non-functional requirements to be considered; *operationalizing softgoals* model lower-level techniques for satisficing NFR softgoals; *claim softgoals* allow the analyst to record design rationale for softgoal refinements, softgoal prioritizations, softgoal contributions, etc. Softgoals can be refined using AND or OR refinements with obvious semantics. Also, softgoal interdependencies can be captured with positive ("+") or negative ("–") contributions.

The main modeling tool that the framework provides is the *softgoal interdependency graph* (SIG). The graphs can graphically represent softgoals, softgoal refinements (AND/OR), softgoal contributions (positive/negative), softgoal operationalizations and claims. As softgoals are being refinement, the developer will eventually reach some softgoals (in fact, operationalizing softgoals) which are sufficiently detailed and cannot be refined further. The developer can accept or reject them as part of the target system. By choosing alternative combinations of the leaf-level softgoals and using the provided label propagation algorithm, the developer can see if the selected alternative is good enough to satisfice the high-level non-functional requirements for the system. The algorithm works its way up the graph starting from the decisions made by the developer. The labelling procedure works towards the top of the graph determining the impact of the decision on higher-level goals. It takes into consideration the labels on softgoal refinement links. For example, if a softgoal receives contributions from a number of other softgoals, then the results of contributions of each offspring are combined to get the overall contribution for the parent softgoal. By analyzing these alternative operationalizations, the developer will select the one that best meets high-level quality requirements for the system. The set of selected leaf level softgoals/operationalizations can be implemented in the software.

The NFR framework also supports cataloguing design knowledge into three main types of catalogues:

- NFR type catalogues include concepts about particular types of NFRs, such as performance.
- Method catalogues encode knowledge that helps in softgoal refinement and operationalization. This catalogue could have a generic method that states that an NFR softgoal applied to a data idem can be decomposed into NFR softgoals for all components of that item.
- Correlation rule catalogues have the knowledge that helps in detecting implicit interdependencies among softgoals. For example, the catalogue could include the fact that indexing positively contributes to response time.

Overall, this framework provides a *process-oriented* approach for dealing with non-functional requirements. Here, instead of evaluation the final product with respect to whether it meets its non-functional requirements, the "emphasis is on trying to rationalize the development process in terms of non-functional requirements" [45]. The provided catalogues are an important aid for a requirements engineer.

## 3.2 *i**/Tropos

*i** [60] is an agent-oriented modeling framework that can be used for requirements engineering, business process reengineering, organizational impact analysis, and software process modeling. Since we are most interested in the application of the framework to modeling systems' requirements, our description of *i** is geared towards requirements engineering. The framework has two main components: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model.

Since *i** supports the modeling activities that take place before the system requirements are formulated, it can be used for both the early and late phases of the requirements engineering process. During the early requirements phase, the *i** framework is used to model the environment of the system-to-be. It facilitates the analysis of the domain by allowing the modeler to diagrammatically represent the stakeholders of the system, their objectives, and their relationships. The analyst can therefore visualize the current processes in the organization and examine the rationale behind these processes. The *i** models developed at this stage help in understanding why a new system is needed. During the late requirements phase, the *i** models are used to propose the new system configurations and the new processes and evaluate them based on how well they meet the functional and non-functional needs of the users.

*i** centers on the notion of *intentional actor* and *intentional dependency*. The actors are described in their organizational setting and have attributes such as goals, abilities, beliefs, and commitments. In *i** models, an actor depends on other actors for the achievement of its goals, the execution of tasks, and the supply of resources, which it cannot achieve, execute, and obtain by itself, or not as cheaply, efficiently, etc. Therefore, each actor can use various opportunities to achieve more by depending on other actors. At the same time, the actor becomes vulnerable if the actors it depends upon do not deliver. Actors are seen as being *strategic* in the sense that they are concerned with the achievement of their objectives and strive to find a balance between their opportunities and vulnerabilities. The actors are used to represent the system's stakeholders as well as the agents of the system-to-be.

Actors can be agents, roles, and positions. *Agents* are concrete actors, systems or humans, with specific capabilities. A *role* is an "abstract actor embodying expectations and responsibilities" [42]. A *position* is a set of *socially recognized* roles typically played by one agent. This division is especially useful when analyzing the social context for software systems.

Dependencies between actors are identified as *intentional* if they appear as a result of agents pursuing their goals. There are four types of dependencies in *i**. They are classified based on the subject of the dependency: *goal*, *softgoal*, *task*, and *resource*.

A Strategic Dependency model is a network of dependency relationships among actors. The SD model captures the intentionality of the processes in the organization, what is important to its participants, while abstracting over all other details. During the late requirements analysis phase, SD models are used to analyze the changes in the organization due to the introduction of the system-to-be. The model allows for the analysis of the direct or indirect dependencies of each actor and exploration of the opportunities and vulnerabilities of actors (analysis of chains of dependencies emanating from actor nodes is helpful for vulnerability analysis).

Strategic Rationale models are used to explore the rationale behind the processes in systems and organizations. In SR models, the rationale behind process configurations can be explicitly described, in terms of process elements, such as goals, softgoals, tasks, and resources, and relationships among them. The model provides a lower-level abstraction to represent the intentional aspects of organizations and systems: while the SD model only looked at the external relationships among actors, the SR model provides the capability to analyze in great detail the internal processes within each actor. The model allows for deeper understanding of what each actor's needs are and how these needs are met; it also enables the analyst to assess possible alternatives in the definition of the processes to better address the concerns of the actors.

The SR process elements are related by two types of links: *decomposition* links, and *means-ends* links. These links are used to model AND and OR decompositions of process elements respectively. Means-ends links are mostly used with goals and specify alternative ways to achieve them. Decomposition links connect a goal/task with its components (subtasks, softgoals, etc.) A softgoal, a goal, or a task can also be related to other softgoals with softgoal contribution links that are similar to the ones in the NFR framework. The links specify two levels of positive ("+" and "++") and negative ("–" and "--") contributions to the softgoals from satisficing a softgoal, achieving a goal, or executing a task. Softgoals are used as selection criteria for choosing the alternative process configuration that best meets the non-functional requirements of the system. It is possible to link a process element from one actor with an intentional dependency going to another actor to represent its delegation to that actor. The SR model is strategic in that its elements are included only if they are considered important enough to affect the achievement of some goal. The same rule also helps with requirements pertinence. The *i** meta-framework describing the semantics and constraints of *i** is described in the language Telos [44]. This language allows for the various types of analysis of i* models (e.g., consistency checking between models) to be performed.

The *i** modeling framework is the basis for Tropos, a requirements-driven agent-oriented development methodology [9]. The Tropos methodology guides the development of agent-based systems from the early requirements analysis through architectural design and detailed design to the implementation. Tropos uses the *i** modeling framework to represent and reason about requirements and system configuration choices. Tropos has an associated formal specification language called Formal Tropos [18] for adding constraints, invariants, pre- and post-conditions capturing more of the subject domain's semantics to the graphical models in the *i** notation. These models can be validated by model-checking.

## 3.3 KAOS

The KAOS methodology is a goal-oriented requirements engineering approach with a rich set of formal analysis techniques. KAOS stands for Knowledge Acquisition in autOmated Specification [14] or Keep All Objects Satisfied [34]. KAOS is described in [34] as a multi-paradigm framework that allows to combine different levels of expression and reasoning: semi-formal for modeling and structuring goals, qualitative for selection among the alternatives, and formal, when needed, for more accurate reasoning. Thus, the KAOS language combines semantic nets [6] for conceptual modeling of goals, assumptions, agents, objects, and operations in the system, and linear-time temporal logic for the specification of goals and objects, as well as state-base specifications for operations. In general, each construct in the KAOS language has a two-level structure: the outer graphical semantic layer where the concept is declared together with its attributes and relationships to other concepts, and the inner formal layer for formally defining the concept.

The ontology of KAOS includes *objects*, which are things of interest in the composite system whose instances may evolve from state to state. Objects can be *entities*, *relationships*, or *events*.

*Operations* are input-output relations over objects. Operation applications define state transitions. Operations are declared by signatures over objects and have pre-, post-, and trigger conditions. KAOS makes a distinction between *domain* pre-/post-conditions for an operation and *desired* pre-/post-conditions for it. The former are indicative and describe what an application of the operation means in the domain (without any prescription as to when the operation must or must not be applied) while the latter are optative and capture additional strengthening of the conditions to ensure that the goals are met [36].

An *agent* is a kind of object that acts as a processor for operations. Agents are active components that can be humans, devices, software, etc. Agents perform operations that are allocated to them. KAOS lets analysts specify which objects are observable or controllable by agents.

A *goal* in KAOS is defined in [29] as a "prescriptive statement of intent about some system whose satisfaction in genera requires the cooperation of some of the agents forming that system". Goals in KAOS may refer to services (functional goals) or to quality of services (non-functional goals). In KAOS, goals are organized in the usual AND/OR refinement-abstraction hierarchies. Goal refinement ends when every subgoal is *realizable* by some individual agent assigned to it. That means the goal must be expressible in terms of conditions that are monitorable and controllable by the agent. The requirement and expectation in KAOS are defined in the usual way – the former being a goal under the responsibility of an agent in the system-to-be and the latter being a goal under the responsibility of an agent in the environment. Goal definition patterns are used for lightweight specification of goals at the modeling layer. These are specified in temporal logic and include patterns such as *achieve*, *cease*, *maintain*, *optimize*, and *avoid*. KAOS also has supports additional types of goals [14]. For example, *satisfaction* goals are functional goals concerned with satisfying agent requests; *information* goals are also functional and are concerned with keeping such agents informed about object states; *accuracy* goals, are non-functional goals that require that the states of software objects accurately reflect the state of the observed/controlled objects in the environment (also discussed in [45]). However, a much richer taxonomy of non-functional goals is presented in [12].

Overall, a KAOS specification is a collection of the following core models:

- *goal model* where goals are represented, and assigned to agents,
- *object model*, which is a UML model that can be derived from formal specifications of goals since they refer to objects or their properties,
- *operation model*, which defines various services to be provided by software agents.

KAOS does not provide a method for evaluating the impact of design decisions on non-functional requirements. However, some variation of the NFR framework and its qualitative analysis approach can be easily integrated into KAOS. Overall, KAOS is a well-developed methodology for goal-oriented requirements analysis that is supplied with solid formal framework. During goal refinement, goal operationalization, obstacle analysis and mitigation, KAOS relies heavily on formal refinement patters that are proven once and for all. Therefore, at every pattern application the user gets an instantiated proof of correctness of the refinement for free.

## 3.4 GBRAM

The emphasis of Goal-Based Requirements Analysis Method (GBRAM) [2][3] is on the initial identification and abstraction of goals from various sources of information. It assumes that no goals have been documented or elicited from stakeholders and thus can use existing diagrams, textual statements, interview transcripts, etc. GBRAM involves the following activities: *goal analysis* and *goal refinement*.

Goal analysis is about the exploration of information sources for goal identification followed by organization and classification of goals. This activity is further divided into *explore* activities that explore the available information, *identify* activities that are about extracting goals and their responsible agents from that information, and *organize* activities that classify and organize the goals according to goal dependency relations. GBRAM distinguishes between achievement and maintenance goals.

Goal refinement concerns the evolution of goals from the moment they are first identified to the moment they are translated into operational requirements for the system specification. This activity is in turn divided into *refine* activities that involve the pruning of the goal set (e.g., by eliminating “synonymous goals”), *elaborate* activities that refer to the process of analyzing the goal set by considering possible goal *obstacles* [50] (behaviours or other goals that prevent or block the achievement of a given goal) and constructing *scenarios* (behavioural descriptions of a system and its environment arising from restricted situations) to uncover hidden goals and requirements, and *operationalize* activities that represent the translation of goals into operational requirements. A *requirement* specifies how a goal should be accomplished by a proposed system. *Constraints* in GBRAM provide additional information regarding requirements that must be met in order for a given goal to be achieved. Constraints are usually identified by searching for temporal connectives, such as “during”, “before”, etc.

One activity that GBRAM requires during the goal refinement phase is the identification of goal precedence. This basically represents the need to identify which goals must be achieved before which other ones. The method suggests asking questions like “What goal(s) must follow this goal?” and so on. Also, another useful method for determining precedence relations is to search for agent dependencies. For example [2], if a supervisor depends on the employee to provide him with a time sheet in order to approve a weekly payment, then there is an agent dependency and it is clear what the goal precedence relation is here. Once goal precedence has been established, tables are produced with goals ordered according to it. This process seems very inefficient given the fact that the *i** notation (which was around at the time when GBRAM was introduced) has an elegant way of capturing inter-actor dependencies, which also allows for an easy detection of the kind of goal precedence relations that GBRAM talks about.

Similar to the other GORE approaches, a system and its environment in GBRAM are represented as a collection of *agents*. Here, agents are defined as entities or processes that seek to achieve goals within an organization or system based on the assumed responsibility for the goals.

The GBRAM approach helps in goal elicitation and refinement by arming practicing requirements engineers with standard questions. For example, one possible question to determine if a goal is a maintenance goal “Is continuous achievement of this goal required?”

In GBRAM, goals, agents, stakeholders, etc. are specified in the textual form in *goal schemas*. Surprisingly, the method does not provide a graphical notation for representing goals, goal refinements, agents, etc. While the method involves, for example, creating precedence relationships among goals, such relationships are much easier perceived when represented graphically.

## 4. Active Areas in GORE Research

In this section, we present an overview of recent research in many important areas that are either part of (e.g., goal elicitation) or are of interest to (e.g. requirements analysis for security and privacy) the Goal-Oriented Requirements Engineering.

## 4.1 Goal Elicitation

Identifying goals is not an easy task. While goals could be explicitly stated by the stakeholders or in the various sources of information available to requirements engineers. However, most frequently goals are implicit and therefore the elicitation process must take place. A preliminary analysis of the current system/organization is an important source of goal identification. This analysis can result in a list of problems and deficiencies that can be precisely formulated. The suggestion of [30] is to negate these formulations, thus producing a first list of goals for the system-to-be to achieve. Goals can also be elicited from available documents, interview transcripts, etc. Here, the suggestion is to look for intentional keywords in the documents [26]. In [2] it is noted that stakeholders tend to express their requirements in terms of operations or actions, rather then goals. So, it makes sense to look for action words such as "schedule" or "reserve" when gathering requirements for a meeting scheduler system.

Once some goals have been identified by the requirements engineer, the aim is usually to refine them into progressively simpler goals until these goals can be easily operationalized and implemented. This process is usually done by asking the HOW questions and refining goals through AND/OR refinements. Many GORE approaches stress that when determining *how* a high-level goal can be refined, one needs to consider alternative ways of refining it to make sure that as many options as possible are explicitly represented in goal models and analyzed with respect to high-level criteria (e.g., [7][23]).

A dual technique to the one describe above is the process of eliciting more abstract goals from those already identify by asking WHY these goals exist. One of the main reasons to elicit more abstract goals is that once a more abstract goal is found, it may be possible to refine it and find its important subgoals that were originally left undetected. Similarly, it may be the case that an originally identified goal amounts to just one alternative refinement of its parent goal. Therefore, the identification and further refinement of higher-level goals leads to more complete goal models and thus to more complete requirements specifications.

Another potential source of goals is scenarios. See Section 4.5 for details.

In the context of software maintenance activities it may be useful to analyze legacy systems in terms of their objectives in order to see whether and how these objectives can be met better with the new technology, how the system can be modified to reflect new needs of its user, how the system can be made more flexible, etc. Here, one could start with the analysis of requirements specification and design documents. However, frequently no documentation is available or the documentation is poorly maintained and does not reflect the state of affairs in legacy system. To this end, an approach is proposed in [63] to produce goal models from the source code of legacy systems through code refactoring and intermediate state-based models. Of course, for that approach to work, the source code must be available.

Yet another approach [40] proposes to take a highly customizable common personal software system (e.g. an email system) and systematically create a goal model representing a refinement of high-level user goals into the configuration options that exist in the software. For each configuration item in the "options" of "preference" menu, questions are asked to determine if that configuration item is an operationalization of some functional user goal or if it contributes to some softgoal. Once a goal behind a configuration item is identified, the method assumes that each value of the item corresponds to the alternative way of achieving the identified goal. Then the softgoals that guide the selection of the alternatives are identified. Later, the identified softgoals are related to each other with contribution links and new more abstract softgoals are possibly identified. Once the goal model is complete, it is possible to talk about the configuration of the system using high-level quality criteria. The goal analysis algorithm of [21] can then be used to infer, given the desired level of satisfising of softgoals, the correct configuration of the software system. This approach can help non-technical users configure their complex software systems easily by selecting the desired level of satisficing for privacy, accuracy, performance, and other qualities. It would be interesting to see if this approach can be useful in helping administrators configure (at least initially) complex enterprise systems such as DBMS's or web servers.

## 4.2 Goal Refinement and Analysis

Goal refinement is the core activity in GORE and thus the different approaches provide a variety of notations, refinement patterns, as well as formal and semi-formal analysis methods to support it. The latter range from informal heuristics [2] to qualitative label propagation algorithms [45][20][21] to temporal logic-based techniques [25][33][36] to probabilistic methods [38].

In providing formal support for goal refinement, KAOS introduces generic refinement patterns that are proven correct and complete and thus are easy to use. Darimont and van Lamsweerde define a number of such patters for goal refinement in [13]. The temporal logic patters such as *milestone-driven* or *case-driven* refinement are defined for propositional formulas (a number of first-order refinement patterns are also proposed).

Letier and van Lamsweerde [38] propose a quantitative approach (in the context of the KAOS framework) to reason about partial goal satisfaction. As it often happens, goals do not need to be satisfied in an absolute sense. A statement "this goal needs to be satisfied in at least 80% of the cases" is an example this phenomenon. Still, different system proposals can have positive or negative effect on such goals. Most analysis techniques, such as [13][37], handle absolute goal satisfaction only. However, the approach in [38] presents a technique for specifying partial degrees of goal satisfaction and for quantifying the impact of different system alternatives on high-level goals that may be satisfied only partially. The authors claim that qualitative techniques (such as the NFR framework) do not provide enough precision for accurate decision support. The simplest way to move from a qualitative approach to a quantitative one would be, for example, to replace the "+"/"–", etc. contributions in the NFR framework with numbers as well as to weight the contribution links. The degree of satisfaction of a goal is then some weighted average of the degrees of satisfaction of its subgoals. However, the authors suggest that these numbers have no physical interpretation in the application domain. It is not clear where they come from. So, the approach tries to base itself on objective criteria (the criteria with some physical interpretation in the domain). Thus, the base of the approach is the model of the partial degree of satisfaction of a goal, which is modeled as one or more *quality variables* (domain-specific, goal-related random variables, e.g. "response time") and zero or more *objective functions*, which are domain-specific, goal related quantities to be maximized or minimized, e.g., "response time within 14 minutes" (to be maximized) with the target value for the system-to-be of 95% and the current value of 80%. The propagation rules (refinement equations) are then used to define how the degree of satisfaction of a goal is determined from the degrees of satisfactions of its subgoals. Refinement equations defining how quality variables of a parent goal relate to those of its subgoals can be complex to write and may need to be defined in terms of probability distribution functions on quality variables. To make the approach simpler, [38] defines quantitative refinement patterns. Here, a probabilistic layer was simply added to the patterns of [13][37]. While an interesting addition to the KAOS toolset, the probabilistic methods of this approach need to be evaluated on real data to determine the level of precision that can be provided by such models. The calculations involved in this approach are rather complex and involve the use of specialized mathematical software, which may limit the applicability of the method to, for example, safety critical projects.

In the NFR framework, goals can be refined along two dimensions, their *type* (e.g., security or integrity) and their *topic* (subject matter), such as "bank account". The framework provides a catalogue of methods for systematic refinement of softgoals. A qualitative label propagation algorithm is used for analysis.

The GBRAM method provides strategies and heuristics for goal refinement. It does not provide an algorithm for formal or semi-formal analysis of goal models or goal refinements.

In *i** and Tropos goals and tasks are refined through means-ends and task decompositions. Additionally, goals, tasks, softgoals, and resources needed for the parent goal or task can be delegated to other actors. A Formal Tropos approach [18] can be used for verifying *i** models. This approach uses a KAOS-inspired formal specification language to formalize the models. Model checking can then be used to verify properties about these models. While being able to produce useful counter examples for properties that do not hold, this approach lacks the extended library of patterns and heuristics for guiding the elaboration of goal models that are the hallmark of the KAOS approach.

Two approaches are presented in [20] and [21] for reasoning with goal models. Both approaches have the same formal setting. The first presents qualitative and numerical axiomatizations for goal models and introduces sound and complete label propagation algorithms for these axiomatizations. These algorithms work from the bottom to the top of goal models, in a fashion similar to the labelling algorithm in the NFR framework. In particular, given a goal model and labels for some of the goals, the algorithm propagates these labels towards the root goals. Here, the goal models have AND and OR decompositions as well as contribution links (a la *i** or the NFR framework) labelled with "+", "++", etc. There is no distinction between functional goal and softgoals, so contribution links may be used with any pair of goals. Also, a new set of contribution links is introduced to capture the relationships where the contribution only takes place if a goal is satisfied. Goals in this framework have 4 possible values – satisfied, denied, partially satisfied, and partially denied. For more fine-grained analysis a quantitative variant of the approach is introduced. Here, the evidence for satisfiability/deniability of a goal is represented as a numerical value. The approach adopts a probabilistic model where the evidence of satisfiability/deniability of a goal is represented as the probability that the goal is satisfied/denied. On the other hand, in [21] the authors attempt to solve a different problem: given a goal model, determine if there is a label assignment for leaf goals that satisfies or denies all root goals. A variation of the approach would find a minimum-cost label assignment that satisfies/denies root goals of the graph provided satisfaction/denial of every goal in the model has a certain unit cost. The approach works by reducing the problems to SAT and minimum-cost SAT for boolean formulas. The above approaches provide a less-powerful, but much simpler alternative analysis method for reasoning about satisfaction/partial satisfaction of goals than [38].

## 4.3 Obstacle Analysis

First-sketch definitions of goals, requirements, and assumptions tend to be over-ideal [33]. They are likely to be violated from time to time due to unanticipated behaviour of agents. The lack of anticipation of exceptional behaviours may result in unrealistic, unachievable and/or incomplete requirements. In KAOS, such exceptional behaviours are captured by formal assertions called *obstacles* to goal satisfaction, which define undesirable behaviour.

Obstacles were first proposed by Potts in [50]. He identified obstacles for a particular goal informally by asking certain questions (GBRAM [3] uses the same approach for handling obstacles). For example: “Can this goal be obstructed, and if so, when?” Additionally heuristics for helping with discovering obstacles are identified. These include looking for potential failures and mistakes, looking for confusions about objects (e.g., can the user invite a person who shouldn’t be invited?), contention for resources, etc. In GBRAM, once an obstacle is identified, a scenario for it must be constructed. Anton notes [2] that while obstacles denote the reason why a goal failed, scenarios denote concrete circumstances under which a goal may fail. Scenarios can, thus, be considered instantiations of goal obstacles. Therefore, in GBRAM, scenarios are used to analyze obstacles. They may help uncover hidden goals or other goal obstacles. Once the obstacles have been identified, however, GBRAM does not provide much guidance on how to deal with them.

On the other hand, KAOS embraced obstacles and provides well-developed methods for detecting and mitigating them [33]. An obstacle to a goal is formally defined as an assertion that is consistent with the domain theory, but the negation of the goal is the logical consequence of the theory consisting of the assertion and the domain theory. Once identified, obstacles can be refined in a way similar to goals (e.g., by AND/OR decomposition). Also, it is important for obstacle refinement to try and identify all the alternative subobstacles to increase the robustness of the system. Obstacles are classified as duals of goals. For example, obstacles for satisfaction goals are called *non-satisfaction* obstacles. The goal of obstacle analysis is KAOS is to anticipate exceptional behaviours in order to derive more complete and realistic requirements. Obstacle analysis helps producing much more robust systems by systematically generating (a) potential ways in which the system might fail to meet its goals and (b) alternative ways of resolving such problems early enough during the requirements elaboration and negotiation phase [34].

It’s interesting to note that the more specific the goal/assumption is, the more specific the obstructing obstacle will be [33]. A high-level goal will produce a high-level obstacle, which will need to be refined in order to identify the precise details of the obstruction. Since eliciting/refining what is not wanted is harder than what is wanted, van Lamsweerde and Letier recommend that obstacles be identified from leaf-level goals.

Obstacles can be formally identified as follows. Given a formal specification for a goal G, calculate the preconditions for obtaining the negation of G from the domain theory. Each obtained precondition is an obstacle. An iterative procedure that at every iteration produces potentially finer obstacles is defined in [33]. For refining obstacles KAOS provides formally proven refinement patterns. The AND/OR refinement of obstacles may be seen as a goal-driven version of fault-tree analysis [39]. Once the obstacles have been refined, it is time to look at the ways to resolve them. In general, this process is called goal *deidealization*. There are a number of possibilities for this. First, a goal that aims at avoiding an identified obstacle can be introduced and refined. Second, it is often the case that obstacles cannot be avoided, so goal restoration will be needed from time to time, and so on. It's important to note that if an obstacle is discovered early in the goal elicitation process, it may turn out to be more severe later [33] (for example, because it could also obstruct some goals elicited later). So, premature decisions with respect to the handling of obstacles may lead to the exclusion of the most appropriate alternatives for mitigating obstacles. In general, while it is possible to elicit and refine obstacles informally in KAOS, the formal approach provides a much higher level of assurance that all the possible exceptional behaviours are captured.

## 4.4 Assigning Goals to Agents

In KAOS, agents are assigned leaf-level goals based on their capabilities. The process is similar in GBRAM. KAOS allows requirements engineers to analyze alternative configurations of the boundary between the system-to-be and its environment through the use of OR responsibility links. Thus, it is possible to compare several system configurations where, for example, more or less functionality is automated. In the end, the responsibility for the achievement of a goal lies with one agent. In GBRAM, on the other hand, it is possible for several agents to be responsible for the same goal at different times.

It is important to note that even though all of the GORE approaches view the composite system comprised of the software-to-be and the environment as a collection of agents, many of these methods (e.g., KAOS and GBRAM) deal with goals in an objective way, from the point of view of a requirements engineer trying to elicit the goals of the combined system, decompose them, and assign terminal goals to agents. While this may be acceptable for a large number of systems, sometimes it makes sense to deal with goals subjectively, from the point of view of the owning agent. This may be useful in a number of settings. Subjective point of view is important, for example, when dealing with complex social systems where agents (both human and intelligent software agents) have different and possibly competing interests, or when analyzing security requirements and trying to model the motivations and actions of hostile agents. Another problem with the above approach is that it assumes that the selection of alternatives in the goal model is done before the goals are assigned to agents. This means all the alternatives configurations possibly represented a goal model are lost and the agents in the composite system are not aware of any variability in the domain. Also, since all the choices are made at the requirements analysis time, the implemented system will only support the single selected alternative, which is likely to lead to fragile systems.

The Tropos approach, on the other hand, advocates keeping goals and alternatives around throughout the development process to produce flexible software systems. Since it is based on the *i** notation, actors in the composite systems do not have to be assigned leaf-level goals. In *i**, actors usually have initial goals that represent their needs in the organization. They can be assigned other goal through intentional dependencies. Goals of an actor are refined into subgoals, subtasks, etc. This refinement is done *from the point of view of the actor*. So, we can view it as subjective goal analysis. For example, if an actor represents an attacker trying to compromise the system, the analysis of goals and goal refinements within that actor is done from the point of view of the attacker. The analysis of alternative goal achievement paths can also be done from the point of view of the owning agent. In addition, these alternatives can be kept in the design and be present at runtime for more flexibility. Further, this subjective analysis better reflects the reality where one agent's leaf-level goal (or task in *i**) is another agent's high-level goal that requires further elaboration.

On the formal analysis side, KAOS, in its usual style, provides a formal approach to refine goals so that they can be assigned to agents, to generate alternative responsibility assignments of goals to agents, and to generate agent interfaces [37]. Agent interfaces are specified in terms of parameters, which are monitored/controlled by agents, and can be established from the formal specification of a goal that is assigned to the agent. Goal specification is normally of the form Pre-/Trigger Condition => Post-Condition. To fulfil its responsibility for the goal the agent must be able to evaluate the goal antecedent and establish the goal consequent. Variables in the antecedent are the monitored parameters, while variables in the consequent are controlled ones.

A goal can only be assigned to an agent that can *realize* it. In order to generate only realizable assignments, [37] proposes a taxonomy of realizability problems and a catalogue of tactics for resolving these problems. For example, some of the realizability problems are the *lack of monitorability* of some variable by the delegated agent, the *lack of controllability* of a variable by the agent, and the *unbounded achievement* goals, where goals do not constrain the behaviour of an agent. The tactics are refinement patterns that help avoid the above-mentioned problems. For instance, a tactics named *Introduce Tracking Object* for solving the lack of monitorability problem calls for maintaining an internal image of the object that cannot be monitored by the delegated agent. While generating alternative assignments of goals to agents, the approach by Letier and van Lamsweerde [37] provides no support for evaluating alternatives and selecting the most promising ones. The NFR framework is suggested by the authors as the possible approach for the analysis of alternative responsibility assignments.

## 4.5 Goals and Scenarios

As mentioned above, stakeholders frequently have problems expressing their needs in terms of goals. They are often much more comfortable talking about potential interactions with the system-to-be to illustrate their expectations from it. Thus, collecting scenarios is a possible way to induce higher-level goals if stakeholders. Moreover, scenarios may lead to a deeper understanding of the system, which in turn makes goal identification easier.

A scenario is defined in [35] as a temporal sequence of interactions among different agents in the restricted context of achieving some implicit purpose. Scenarios are used in requirements engineering (e.g., [58][50][2]) and proved to be useful in requirements elicitation, obstacle analysis and mitigation, etc. Scenarios have a number of strengths: they are informal, narrative, and concrete descriptions of hypothetical interactions between the software and its environment. They are usually easily comprehensible by stakeholders. However, requirements are inherently partial. They give rise to the coverage problem (similar to testing) making it impossible to verify the absence of errors. Also, enumerating combinations of individual behaviours leads to combinatorial explosion [26]. It is only possible to induce a specification from a collection of scenarios. Being procedural, scenarios may result in over-specification. Also, scenarios leave required properties of the intended system implicit [35]. Nevertheless, a number of goal-based requirements analysis approaches embraced scenarios. Scenarios can be used for initial goal elicitation [35], for helping to uncover obstacles to goal satisfaction [2][50], and to verify system specifications: as [26] points out, scenarios can be very helpful in "deficiency-driven requirements elaboration". Here, a system is specified by a set of goals (usually formalized in some kind of temporal logic) and a set of scenarios. Then, the aim is to detect inconsistencies between scenarios and goals and to modify the specifications if inconsistencies are found. The inconsistency detection can be done by a planner or a model checker. One example of this approach is Formal Tropos [18].

There are a number of approaches the employ scenarios as well as goals. For Potts [50], scenarios are derived from the description of the system's and the user's goals, and the potential obstacles that block those goals. He proposes a scenario schema and a method for deriving a set of *salient* scenarios, scenarios that have a point and help people understand the exemplified system. The idea of scenario salience is thus to limit the number of generated scenarios. The goal of the approach is to use scenarios to understand user needs. According to Potts, this process must precede the writing of a specification for a system. In the proposed scenario schema, each *episode* (a major chunk of activity in a scenario) corresponds directly to a domain goal. The goal-scenario process of Potts is interleaving scenario derivation and goal refinement. Scenarios can be obtained by analyzing goals, goal allocations, obstacles, etc. On the other hand, scenarios may give the analyst an insight about goals and goal refinement. Potts suggests that salient scenarios are beneficial in that they can be used to compare alternative system proposals, including more or less automation, various alternative ways of achieving goals, etc. Also, scenarios can be employed to help in mitigating the effects of obstacles. While it is conceivable to assume that scenarios generated for several system alternatives could occasionally illustrate the benefit of one alternative over another, this approach for evaluation of alternatives does not seem systematic compared to, say, the NFR framework.

Another goal-scenario approach is described by Rolland et al. [52]. The CREWS-L'Ecritoire creates a bi-directional coupling between goals and scenarios. The proposed process has two parts. When a when a goal is discovered, a scenario can be authored for it and once a scenario has been authored, it is analysed to yield goals. By exploiting the goal-scenario relationship in the reverse direction, i.e. from scenario to goals, the approach pro-actively guides the requirements elicitation process. The authors introduce *requirement chunks*, which are pairs of goals and scenarios. Scenarios are composed of actions, an action being an interaction of agents. Scenarios can be normal and exceptions. The former lead to the achievement of the associated goal, while the latter lead to its failure.

Goals in CREWS are encoded textually. Goal descriptions in this approach are quite complex and are expressed as a clause with the main verb and a number of parameters including quality, object, source, beneficiary, etc. In this process, goal discovery and scenario authoring are complementary steps and goals are incrementally discovered by repeating the goal-discovery, scenario-authoring cycle. Goal discovery in this method is on the linguistic analysis of goal statements.

Requirement chunks can be composed using AND/OR relationship or through refinement, which relates chunks of at different levels of abstraction.

The approach can generate a very large number of alternative ways to achieve a goal by iterating through all the possible values for all the possible parameters of the goal. This large number of alternative goals must be pruned to remove meaningless goals. The way to do it is to create a scenario for it and to test whether the goal is *realistic*. While it may seem strange that potential goal refinements are generated purely linguistically, the authors claim that their approach works since in the context of RE, the larger the number of alternatives explored the better. The CREWS experience paper [53] claims that the automatic generation of potential alternatives fared better than ad hoc methods.

In the context of the KAOS approach, van Lamsweerde and Willemet [35] proposed a formal approach to infer specifications of system goals and requirements inductively from interaction scenarios. To do this, the method takes scenarios represented as UML sequence diagrams as examples/counterexamples of intended system behaviour, and inductively infers a set of candidate goals/requirements that cover all example scenarios and exclude all counterexample scenarios. The idea here is to generate temporal logic formulas whose logical models include/exclude the temporal sequences given as positive/negative scenarios. The method is based on a learning algorithm.

Since inductive inference is not sound, the requirements engineer, thus, must check the candidate assertions for adequacy. Similar to Rolland et al., van Lamsweerde and Willemet state that goal elaboration and scenario elaboration are intertwined processes; a concrete scenario description may prompt the elicitation of the specifications of the goals underlying it and a goal specification may prompt the elaboration of scenario descriptions to illustrate or validate it. The objective of goal-scenario integration in KAOS is to obtain from scenarios additional goal specifications that could not be obtained from the goal refinement/abstraction process that began with the goals initially identified from interviews or existing documentation. The additional goals obtained from scenarios may be brand new; they may cover specifications already found by the goal elaboration process (in which case the elicitation just reduces to some form of validation); they may also be conflicting with specifications found by the goal elaboration process [35].

## 4.6 Handling Goal Conflicts in GORE

Different stakeholders (clients, users, requirements engineers, developers, etc.) in general have different objectives, needs, concerns, perceptions, knowledge, and skills. In order to produce an adequate and complete requirements specification, all relevant viewpoints [16] on the system need to be captured and integrated, with their differences resolved appropriately. The importance of viewpoints has been recognized since the early days of requirements engineering [55]. While inconsistencies may be sources of new information, eventually, they need to be resolved.

It was noted by Robinson [51] that the roots of many inconsistencies in requirements engineering are conflicting goals and therefore the level of goals is where conflicts can be detected and resolved. He proposed to capture views of system stakeholders and due to the usual egocentricity of those views, Robinson called them *selfish views*. The combination of such views will most likely involve resolving conflicts among the goals of the stakeholders. Attributes such as importance, utility, feasibility, etc. are associated with goals. The use of analytic decision technique that uses utility functions is proposed.

A goal-based approach for requirements negotiation was offered by Boehm et al. [5]. It proposes a model where all stakeholders are identified together with their *win conditions* (goals); conflicts between their goals are then captured together with their associated risks and uncertainties; after that, goals are reconciled through negotiation to rich a mutually agreeable set of goals. The proposed process, as any negotiation process, is iterative.

In [33] and [25], van Lamsweerde and Letier proposed an approach to capture different views during requirements analysis and to detect and resolve conflicts among them in the context of the KAOS framework. This method is much more formal than the approaches in [5] and [51]. They define views as ternary relationships linking an actor, a master concept (e.g., a goal, an object, an agent, an operation, etc.), and a facet of it. Views associated with the same actor can be grouped together to form *perspectives*. Views are said to be *overlapping* if their respective facets share an attribute, link, or predicate. The approach categorizes various possible inconsistencies among overlapping views. Among them are *conflicts* and *divergences*. Conflicts are situations where assertions, which formalize goals, assumptions, or requirements, are inconsistent in the domain theory. A weaker form of conflict, called divergence, occurs if there exists some boundary condition (that can be established through some scenario) that makes the otherwise consistent assertions inconsistent if conjoined to them. Variations of divergences, *obstructions* and *competitions*, are also defined. A formal method for detecting conflicts by either regressing negated assertions of by using the divergence patterns is offered in [33]. Pattern-based resolution options are also defined. While offering a solid formal framework for dealing with conflicts (including the patterns and heuristics that are present in every aspect of the KAOS framework), this approach fails to take into consideration the social aspect of conflicts. For example, it does not consider goal priority or the importance of agents in the organization to resolve conflicts. Non-functional requirements are not considered either.

## 4.7 Capturing Variability in GORE

Coming up with a solid (complete, correct, etc.) requirements specification for a software system involves the identification of the many alternative ways to achieve goals, assign goals to agents, draw a system-environment boundary, and so on, as well as making the choice among the identified alternatives. Various GORE approaches support this activity differently. For example, one approach, the Goals-Skills-Preferences framework of [23] aims at designing highly-customizable software systems by discovering as much variability in the problem domain as possible. Here, given high-level user goals the framework attempts to identify *all* the possible ways these goals can be achieved by the combined system. The variability is modeled by the OR decompositions of goals in the usual AND/OR goal graph. A ranking algorithm for selecting the best system configuration among this vast space of alternatives is proposed. It takes into consideration user *preferences*, which are modeled as softgoals and are accompanied by the contribution links relating them to the functional goals, and user *skills*, which are hard constraints on the leaf-level goals. Each user is defined through a skills profile – an evaluation of the user's skills such as vision, speech production, etc. Each leaf-level goal, such as "dictate a letter", needs a certain set of skills from the user to be achieved. It is clear, that the "dictate a letter" goal needs the skill of speech production at a reasonable level. So, the for each particular user, the algorithm can prune the alternatives that are incompatible with the user's skills and then select the best one from the remaining set according to the user preferences. The result is a specification that is tailored for a concrete user.

In Tropos and other *i**-based approaches, variability is captured using means-ends or OR decompositions. The modeling of the alternative responsibility assignments for goals, tasks, softgoals, and resources is also supported. For instance, one can model a number of alternative ways of achieving a certain goal. One is to let the actor that owns the goal achieve it, another is to delegate the goal to another actor through an intentional goal dependency, and the third may be to delegate it to a yet another actor. The same approach can be used to analyze the system-environment boundary. By delegating more goals to the actors who are part of the system-to-be we are modeling the situation where more functionality of the combined system is automated. On the other hand, delegating the achievement of more goals to the actors in the environment will shift the balance in an opposite direction. These alternatives can be analyzed through their contributions to the high-level quality requirements modeled as softgoals. The standard softgoal analysis method based on the NFR framework is used.

In KAOS, variability can be represented using OR goal decompositions and alternative responsibility assignments of goals to agents. Additionally, identified obstacles can be resolved differently by alternative decompositions and application of alternative obstacle resolution patterns. Similarly, various heuristics can be applied to conflict resolution in KAOS. All of these activities can result in alternative system proposals. While KAOS does not provide any integrated tools for qualitative analysis of different alternatives with respect to quality requirements, the use of the NFR framework is often suggested. A quantitative approach that can be to reasoning about system alternatives is proposed in [38].

The CREWS approach [52] is able to automatically generate a very large number of potential goal refinements by modifying the parameters of the textual representation for goals (see Section 4.5). These are validated by scenarios. The approach does not suggest a way to systematically analyze alternative system configurations.

## 4.8 From Requirements to Architectures

Architectural design has long been recognized as having a major impact on non-functional requirements of systems [49]. However, systematic approaches to build software architectures that satisfy systems' functional and non-functional requirements are rare. In the context of GORE, there are a number of proposals to methodically build software architectures based on functional and non-functional requirements.

One such approach is presented in [11]. It proposes to use the NFR framework to reason about the effects of architectural decisions on the qualities of the system under development and to use the *i** Strategic Dependency models to represent the social/organizational context for the system. The NFR framework allows for relating architectural decisions to softgoals through appropriately labelled contribution links. Chung et al. suggest making the information about the relative criticality of softgoals explicit in the model in order to help with the tradeoff analysis among the softgoals. Also, *claims* should be used in the NFR model to justify softgoal modeling decisions. By using the softgoal labelling algorithm, the architect could easily see how architectural decisions affect the quality requirements of the system. In addition, the approach suggests using SD diagrams to model stakeholders, their quality requirements, and their dependencies. In these models, the stakeholders and the architect(s) are explicitly modeled with the architect being delegated the softgoals such as extensibility, good time performance, etc. This way, SD models represent the source of each non-functional requirement for the system. This approach mainly concentrates on the evaluation of architectural decisions. There is no support for systematic derivation of software architecture based on the desired qualities of the system.

An *i**-based proposal for handling the evolution of system architectures based on changing business goals is described in [22]. It proposes to systematically relate business goals (captured as softgoals) to architectural design decisions and architectural structures during the development and evolution of software systems.

Another approach [41] proposes to use the *i**-based Goal-oriented Requirements Language (GRL) together with Use Case Maps [8], which provide a way to visualize scenarios using *scenario paths* and superimpose them on models representing the structure of abstract components. In this approach, GRL is used to support goal and agent-oriented modeling and reasoning and to guide the architectural design process. UCM notation is used to express the architectural design at each stage of the development. GRL provides support for reasoning about scenarios by establishing correspondences between intentional elements in GRL and functional components and *responsibilities* (tasks) in UCM. The iterative process uses GRL to reason about how architectural decisions affect high-level non-functional requirements (using the usual label propagation algorithm), while UCM is used to generate and analyze how responsibilities can be bound to architectural components and the effects of these bindings. As new architectural alternatives are proposed during the UCM analysis, they are analyzed in GRL with respect to their contributions to the quality requirements. It is interesting to note that in the this approach, GRL models have intentional elements that do not model goals/tasks of the actors that are part of the combined system, but represent possible architectural decisions (e.g., put component A into device B) and their impact on non-functional requirements.

In the context of the KAOS approach, van Lamsweerde [28] proposes an approach for software architecture design from KAOS goal models. The starting point for the method is the software specification that can be systematically produced from software requirements in KAOS [36]. Then, components are created for agents that are assigned to achieving goals of the system-to-be. Component interfaces are derived based on the sets of variables the agents control and monitor. A dataflow connector is created for each combination of two components where one controls a variable, which the other agent monitors. This produces an initial architectural view for the system. This model is then restructured according to the desired architectural style. Another pattern-based restructuring follows. This time the aim is to improve the quality of service goals. The interesting aspect of this approach is that it generates the initial architectural model directly from the goal model, using the precedence relationships among goals to create data connectors. This means that the initial software architecture can be automatically generated from a goal model. Thus, it will be quite interesting to investigate whether instead of refining the architecture to meet quality requirements one could apply refinement patterns to goal models themselves.

## 4.9 Analyzing Security Requirements

As more and more business processes in the world are being automated and more and more sensitive data needs to be accessible by these processes the need for solid approaches for elicitation and analysis of security and privacy requirements. A number of extensions to the GORE approaches for handling security requirements have emerged recently.

An addition to the KAOS framework for handling security requirements is proposed in [29]. It proposes the use of patterns to identify security goals of the system-to-be. By negating these goals, one gets *anti-goals* – the goals of potential attacker. For each anti-goal, potential owners (attackers) are identified and their higher-level anti-goals are elicited. The anti-models are then further refined and operationalized. A set of tactics to develop countermeasures is proposed (e.g., avoid vulnerability, goal restoration) to be applied to the system model. The formal language is extended with epistemic constructs for reasoning about attackers' knowledge. Thus, the approach allows for explicit identification of attacker agents and explicit modeling of their goals.

An *i**-based approach for handling security and privacy requirements is introduced by Liu, Yu, and Mylopoulos in [42]. It aims at providing extensive support for modeling of the social context of software systems. The method starts with a plain *i** model for the combined system. Then each actor in the model is considered in turn as a potential attacker. For every attacker the analyst then identifies malicious intents. Since the attacker is an insider, it has access to all resources, relationships, etc. of its legitimate counterpart. Then, the dependency links are used to determine whether/how other actors in the system can become vulnerable by depending on an actor who now is considered being an attacker. To help prevent the vulnerabilities the authors suggest applying pattern-based solutions (e.g., from [12]) to address the identified problems. The selection among alternatives is done as usual with respect to the identified softgoals using the well-known label propagation algorithm. The approach only supports reasoning about insider attackers only, which can be seen as a drawback. In [61], an opposite approach was adopted, where an existence of a trusted perimeter was assumed for each actor and only threats from the outside of it were analyzed.

A policy-based extension to GBRAM for handling security and privacy requirements is proposed in [4]. This approach suggests heuristics that aim at helping practitioners with the identification and formulation of policies that can be operationalized into requirements.

### 4.10 Requirements Monitoring

A KAOS-based approach for monitoring and resolving requirements violations at runtime is suggested in [17]. It proposes to identify breakable assertions in the specification and to select the ones that require a monitor assigned to them for runtime monitoring. Monitoring parameters are then identified for each breakable assertion and appropriate thresholds need to be defined for these parameters. A reconciliation tactics need to be identified for each breakable assertion. The first option is to introduce a restoration procedure for an assertion violation. The other option is to choose an alternative course of actions to achieve the same parent goal. Of course, to be able to change the behaviour dynamically, multiple alternative system configurations must be available at runtime. A useful modification of this approach would be to make it more aware of the system's non-functional requirements. For example, when shifting to an alternative design to reconcile the runtime behaviour and the requirements, if a number of alternatives are available, they should be selected based on their contribution to the system's quality requirements.

## 5. Open Problems

In this section we would like to briefly discuss a number of problems in the area of Goal-Oriented Requirements Engineering that require are still open and are of interest to us.

1. An important area of research for us is the representation and analysis of alternative system proposals/configurations using goal-based techniques. Whether the goal is to create a highly customizable system [23], an adaptive system, or to analyze the alternatives thoroughly at requirements time and select a single configuration for implementation, we need means to capture and analyze variability in the problem domain. In [23], goal models were shown to be useful for capturing such variability and a Goals-Skills-Preferences (GSP) framework was sketched for the analysis of alternatives in the domain of personal software. However, it is clear that fine-grain representation of user skills is all but irrelevant in the domain of enterprise software: while a large variety of people with vastly varying skill sets are using personal software at home and would like to improve their user experience by customizing the software based on their skills and preferences, in an enterprise the goals are different. End users matter less than the efficient business processes. Therefore, analyzing alternative configurations for an enterprise system, we need a different analysis framework. The framework will still rely on goals. It will probably rely on preferences, which now represent not the personal preferences of the user, but high-level business goals. On the other hand, the Skills (the hard constraints on alternatives in the GSP framework) will need to be replaced by domain-specific metrics. The same will happen in other domains. Therefore, the development of some kind a meta-framework for the analysis of alternatives based on goals, softgoals, hard constraints, and possibly other criteria, which can be instantiated to GSP-like analysis frameworks in any domain would quite beneficial. The problem of the selection of the best alternative remains largely open.

2. Another interesting problem is the manipulation of systems through their goal models. For example, merging of several goal models could correspond to the Enterprise Application Integration; addition, removal or replacement or goals may represent requirements evolution, etc. Some of the existing techniques (e.g., approaches for handling goal conflicts) may help in this.

3. While GORE and agent-oriented software engineering are a nice match given the fact that both have goals as a central concept, there is still much to do in the area of developing RE approaches for truly open and dynamic multiagent systems populated by smart deliberating agents capable of constructing forming teams and constructing plans dynamically. How can we estimate whether the behaviour of such system will meet its requirements? How can we analyze alternatives for goal achievement if agents are able to do planning at runtime? It seems that in this situation in order to guarantee that the requirements are met, we need to use the same tools for requirements enforcement as the government uses in dealing with the population: laws, policies, law enforcement, etc. This will hopefully provide the freedom for agents while guaranteeing some high-level properties of the system.

## 6. Bibliography

[1] A. Anton, W. McCracken, C. Potts. Goal Decomposition and Scenario Analysis in Business Process Reengineering. Proc. 6th Conference On Advanced Information Systems Engineering (CAiSE'94), Utrecht, Holland, June 1994.

[2] A. Anton. Goal-Based Requirements Analysis. Proc. Second IEEE International Conference on Requirements Engineering (ICRE'96), Colorado Springs, USA, April 1996.

[3] A. Anton. Goal Identification and Refinement in the Specification of Software-Based Information Systems. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, USA, June 1997.

[4] A. Anton, J. Earp, A. Reese. Analyzing Website Privacy Requirements Using a Privacy Goal Taxonomy. Proc. Joint International Requirements Engineering Conference (RE'02). Essen, Germany, September 2002.

[5] B.W. Boehm, P. Bose, E. Horowitz, M. Lee. Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach. Proc. 17th International Conference on Software Engineering (ICSE'95), Seattle, USA, April 1995.

[6] R. Brachman, H. Levesque (Eds.). Readings in Knowledge Representation. Morgan Kaufmann, 1985.

[7] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent Systems, 8(3), May 2004.

[8] R. Buhr, R. Casselman. Use Case Maps for Object-Oriented Systems. Prentice Hall, 1996.

[9] J. Castro, M. Kolp, J. Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems, 27(6), September 2002.

[10] P. Checkland, S. Holwell. Information, Systems and Information Systems – Making Sense of the Field. Wiley. 1998.

[11] L. Chung, D. Gross, E. Yu. Architectural Design to Meet Stakeholder Requirements. In P. Donohue (Ed.), Software Architecture. Kluwer, 1999.

[12] L. Chung, B. Nixon, E. Yu, J. Mylopoulos. Non-Functional Requirements in Software Engineering. Kluwer. 2000.

[13] R. Darimont, A. van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. Proc. 4th Symposium on the Foundations of Software Engineering (FSE-4), San Francisco, USA, October 1996.

[14] A. Dardenne, A. van Lamsweerde and S. Fickas. Goal-Directed Requirements Acquisition. Science of Computer Programming, 20(1-2), April 1993.

[15] T. DeMarco. Structured Analysis and System Specification. Yourdon Press. 1978.

[16] S. M. Easterbrook. Domain Modelling with Hierarchies of Alternative Viewpoints. Proc. 1st IEEE International Symposium on Requirements Engineering (RE'93), San Diego, January 1993.

[17] M. Feather, S. Fickas, A. van Lamsweerde, C. Ponsard. Reconciling System Requirements and Runtime Behaviour. Proc. 9th International Workshop on Software Specification and Design (IWSSD'98), Ise-Shima, Japan, April 1998.

[18] A. Fuxman, M. Pistore, J. Mylopoulos, P. Traverso. Model Checking Early Requirements Specifications in Tropos. Proc. 5th International Symposium on Requirements Engineering (RE'01), Toronto, Canada, August 2001.

[19] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, P. Traverso. Specifying and analyzing early requirements in Tropos. Requirements Engineering 9(2), May 2004.

[20] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In LNCS 2503, 2002.

[21] P. Giorgini, J. Mylopoulos, R. Sebastiani. Simple and Minimum-Cost Satisfiability for Goal Models. Proc. 16th Conference On Advanced Information Systems Engineering (CAiSE'04), Riga, Latvia, June 2004.

[22] D. Gross, E. Yu. Evolving System Architecture to Meet Changing Business Goals: an Agent and Goal-Oriented Approach. Proc. Workshop From Software Requirements to Architectures (STRAW'2001), Toronto, Canada, May 2001.

[23] B. Hui, S. Liaskos, J. Mylopoulos. Requirements Analysis for Customizable Software: A Goals-Skills-Preferences Framework. Proc. International Conference on Requirements Engineering (RE'03), Monterey, USA, September 2003.

[24] M. Jackson. The Meaning of Requirements. Annals of Software Engineering vol.3, 1997.

[25] A. van Lamsweerde. Divergent Views in Goal-Driven Requirements Engineering. Proc. Workshop on Viewpoints in Software Development, San Francisco, USA, October 1996.

[26] A. van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. 22nd International Conference on Software Engineering (ICSE'2000), Limerick, Ireland, June 2000.

[27] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. Proc. 5th IEEE International Symposium on Requirements Engineering (RE'01), Toronto, Canada, August 2001.

[28] A. van Lamsweerde. From System Goals to Software Architecture. In M. Bernardo & P. Inverardi (Eds), Formal Methods for Software Architectures, LNCS 2804, Springer-Verlag, 2003.

[29] A. van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. Proc. 26th International Conference on Software Engineering (ICSE’04), Edinburgh, UK, May 2004.

[30] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. Proc. International Conference on Requirements Engineering (RE’04), Kyoto, Japan, September 2004.

[31] A. van Lamsweerde, R. Darimont, P. Massonet. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. Proc. Second International Conference on Requirements Engineering (RE’95), York, UK, March 1995.

[32] A. van Lamsweerde, R. Darimont, E. Letier. Managing Conflicts in Goal-Oriented Requirements Engineering. IEEE Transactions on Software Engineering, 24(11), November 1998.

[33] A. van Lamsweerde, E. Letier. Handling Obstacles in Goal-Oriented Software Engineering. IEEE Transactions on Software Engineering, 26(10), October 2000.

[34] A. van Lamsweerde, E. Letier. From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering. Proc. Radical Innovations of Software and Systems Engineering, LNCS, 2003.

[35] A. van Lamsweerde, L. Willemet. Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Transactions on Software Engineering, 24(12), December 1998.

[36] E. Letier, A. van Lamsweerde. Deriving Operational Software Specifications from System Goals. Proc. 10th Symposium on the Foundations of Software Engineering (FSE-10), Charleston, USA, November 2002.

[37] E. Letier, A. van Lamsweerde. Agent-Based Tactics for Goal-Oriented Requirements Elaboration. Proc. 24th International Conference on Software Engineering (ICSE’02), Orlando, USA, May 2002.

[38] E. Letier, A. van Lamsweerde. Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering. Proc. 12th ACM International Symposium on the Foundations of Software Engineering (FSE’04), Newport Beach, USA, November 2004.

[39] N. Leveson. Safeware – System Safety and Computers. Addison-Wesley. 1995.

[40] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, S.Easterbrook. Configuring Common Personal Software: a Requirements-Driven Approach. Proc. 13th International Requirements Engineering Conference (RE’05), Paris, France, August 2005.

[41] L. Liu, E. Yu. From Requirements to Architectural Design – Using Goals and Scenarios. Proc. Workshop From Software Requirements to Architectures (STRAW'2001), Toronto, Canada, May 2001.

[42] L. Liu, E. Yu, J. Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting. Proc. International Conference on Requirements Engineering (RE'03), Monterey, USA, September 2003.

[43] G. Kotonya, I. Sommerville. Requirements Engineering: Processes and Techniques. Wiley, 1998.

[44] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis. Telos: Representing Knowledge About Information Systems. ACM Transactions on Information Systems, 8(4), October 1990.

[45] J. Mylopoulos, L. Chung, B. Nixon. Representing and Using Non-Functional Requirements: A Process-Oriented Approach. IEEE Transactions on Software Engineering, 18(6), June 1992.

[46] N. Nilsson. Problem Solving Methods in Artificial Intelligence. McGraw Hill, 1971.

[47] B. Nuseibeh, S. Easterbrook. Requirements Engineering: A Roadmap. Proc. Conference on the Future of Software Engineering, Limerick, Ireland, June 2000.

[48] D. Parnas, J. Madey. Functional Documents for Computer Systems. Science of Computer Programming, Vol. 25, 1995.

[49] D. Perry, A. Wolf. Foundations for the Study of Software Architecture. ACM Software Engineering Notes, 17(4), October 1992.

[50] C. Potts. Using Schematic Scenarios to Understand User Needs. Proc. Designing Interactive Systems (DIS'95), Ann Arbor, USA, August 1995.

[51] W. Robinson. Integrating Multiple Specifications Using Domain Goals. Proc. 5th International Workshop on Software Specification and Design (IWSSD-5), Pittsburgh, USA, May 1989.

[52] C. Rolland, C. Souveyet, C. Ben Acour. Guiding Goal Modeling Using Scenarios. IEEE Transactions on Software Engineering, 24(12), December 1998.

[53] C. Rolland, G. Grosz, R. Kla. Experience with Goal-Scenario Coupling in Requirements Engineering. Proc. 4th International Symposium on Requirements Engineering (RE'99), Limerick, Ireland, June 1999.

[54] D. Ross. Structured Analysis: A Language for Communicating Ideas. IEEE Transactions on Software Engineering, 3(1), January 1977.

[55] D. Ross, K. Schoman. Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering, 3(1), January 1977.

[56] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. Object-Oriented Modeling and Design. Prentice Hall. 1991.

[57] H. Simon. The Sciences of the Artificial, 2nd Ed. MIT Press, 1981.

[58] A. Sutcliffe, N. Maiden. Supporting Scenario-Based Requirements Engineering. IEEE Transactions on Software Engineering, 28(12), December 1998.

[59] E. Yu. Why Agent-Oriented Requirements Engineering. Proc. 3rd International Workshop on Requirements Engineering: Foundations for Software Quality, Barcelona, Spain, June 1997.

[60] E. Yu. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. Proc. 3rd International Symposium on Requirements Engineering (RE'97), Washington, USA, January 1997.

[61] E. Yu, L. Liu. Modeling Trust for System Design Using the *i** Strategic Actors Framework. In R. Falcone, M. Singh, Y.H. Tan (Eds.), Trust in Cyber-Societies – Integrating the Human and Artificial Perspectives. LNAI-2246, 2001.

[62] E. Yu, J. Mylopoulos. Why Goal-Oriented Requirements Engineering. Proc. 4th International Workshop on Requirements Engineering: Foundations of Software Quality, Pisa, Italy, June 1998.

[63] Y. Yu, Y.Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian, J.C. Leite. Refactoring Source Code Into Goal Models. Proc. 13th International Requirements Engineering Conference (RE'05), Paris, France, August 2005.

[64] K. Yue. What Does It Mean to Say that a Specification is Complete? Proc. Fourth International Workshop on Software Specification and Design (IWSSD-4), Monterey, USA, 1987.

[65] P. Zave. Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys, 29(4), 1997.