Goal-Oriented Requirements Engineering: A Guided Tour

Axel van Lamsweerde

Département d'Ingénierie Informatique
Université catholique de Louvain
B-1348 Louvain-la-Neuve (Belgium)
avl@info.ucl.ac.be

Abstract

Goals capture, at different levels of abstraction, the various objectives the system under consideration should achieve. Goal-oriented requirements engineering is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements. This area has received increasing attention over the past few years.

The paper reviews various research efforts undertaken along this line of research. The arguments in favor of goal orientation are first briefly discussed. The paper then compares the main approaches to goal modeling, goal specification and goal-based reasoning in the many activities of the requirements engineering process. To make the discussion more concrete, a real case study is used to suggest what a goal-oriented requirements engineering method may look like. Experience with such approaches and tool support are briefly discussed as well.

1. Introduction

Goals have long been recognized to be essential components involved in the requirements engineering (RE) process. As Ross and Schoman stated in their seminal paper, *"requirements definition must say why a system is needed, based on current or foreseen conditions, which may be internal operations or an 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"* [Ros77]. Many informal system development methodologies from the good old times included some form of goal-based analysis, called context analysiis [Ros77], definition study [Hic74], participative analysis [Mun81], and so forth. Typically, the current system under consideration is analyzed in its organizational, operational and technical setting; problems are pointed out and opportunities are identified; high-level goals are then identified and refined to address such problems and meet the opportunities; requirements are then elaborated to meet those goals. Such natural practice has led requirements documentation standards to require a specific document section devoted to the objectives the system should meet (see, e.g., the IEEE-Std-830/1993 standards).

Surprisingly enough, goals have been largely ignored both from the literature on software modeling and specification and from the literature on object-oriented analysis (one notable exception is [Rub92]). UML advocates sometimes confess the need for higher-level abstractions: *"In my work, I focus on user goals first, and then I come up with use cases to satisfy them; by the end of the elaboration period, I expect to have at least one set of system interaction use cases for each user goal I have identified"* [Fow97, p.45]). The prominent tendency in software modeling research has been to abstract programming constructs up to requirements level rather than propagate requirements abstractions down to programming level [Myl99].

Requirements engineering research has increasingly recognized the leading role played by goals in the RE process [Yue87, Rob89, Ber91, Dar91, Myl92, Jar93, Zav97b]. Such recognition has led to a whole stream of research on goal modeling, goal specification, and goal-based reasoning for multiple purposes, such as requirements elaboration, verification or conflict management, and under multiple forms, from informal to qualitative to formal.

The objective of this paper is to provide a brief but hopefully comprehensive review of the major efforts undertaken along this line of research. Section 2 first provides some background material on what goals are, what they are useful for, where they are coming from, and when they should be made explicit in the RE process. Section 3 discusses the major efforts in modeling goals in terms of features and links to other artefacts found in requirements models. Section 4 reviews the major techniques used for specifying goals. Section 5 on goal-based reasoning reviews how goals are used in basic activities of the RE process such as requirements elicitation, elaboration, verification, validation, explanation, and negotiation, and in particular for difficult aspects of that process such as conflict management, requirements deidealization, and alternative selection. Section 6 then suggests what a goal-oriented RE method may look like by enacting it on a real case study of a safety-critical train control system. This naturally leads to a brief review, in Section 7, of industrial projects in which the use of such methods was felt conclusive; the supporting tools used in those projects are also briefly discussed there. Section 8 just opens some fairly recent pieces of goal-based work beyong requirements engineering.

2. The background picture

Reviewing the current state of the art in goal-oriented RE would not make much sense without first addressing the what, why, where and when questions about this area of research.

Invited mini-tutorial paper, appeared in
Proceedings RE'01, 5th IEEE International Symposium on Requirements Engineering, Toronto, August 2001, 249-263.

What are goals?

A *goal* is an objective the system under consideration should achieve. Goal formulations thus refer to intended properties to be ensured; they are optative statements as opposed to indicative ones, and bounded by the subject matter [Jac95, Zav97a].

Goals may be formulated at different levels of abstraction, ranging from high-level, strategic concerns (such as *"serve more passengers"* for a train transportation system or *"provide ubiquitous cash service"* for an ATM network system) to low-level, technical concerns (such as *"acceleration command delivered on time"* for a train transportation system or *"card kept after 3 wrong password entries"* for an ATM system).

Goals also cover different types of concerns: functional concerns associated with the services to be provided, and non-functional concerns associated with quality of service --such as safety, security, accuracy, performance, and so forth.

The *system* which a goal refers to may be the current one or the system-to-be; both of them are involved in the RE process. High-level goals often refer to both systems. The system-to-be is in essence composite; it comprises both the software and its environment, and is. made of active components such as humans, devices and software. As opposed to passive ones, active components have choice of behavior [Fea87, Yue87, Fic92]; henceforth we will call them *agents*. Unlike requirements, a goal may in general require the cooperation of a hybrid combination of multiple agents to achieve it [Dar93]. In a train transportation system, for example, the high-level goal of safe transportation will typically require the cooperation of on board train controllers, the train tracking system, station computers, the communication infrastructure, passengers, and so forth. In an ATM system, the goal of providing cash to eligible users will require the cooperation of the ATM software, sensors/actuators, the customer, etc. One of the important outcomes of the RE process is the decision on what parts of the system will be automated and what parts will not. A goal under responsibility of a single agent in the software-to-be becomes a *requirement* whereas a goal under responsibility of a single agent in the environment of the software-to-be becomes an *assumption* [Lam98b, Lam98c]. Unlike requirements, assumptions cannot be enforced by the software-to-be; they will hopefully be satisfied thanks to organizational norms and regulations, physical laws, etc.

Why are goals needed?

There are many reasons why goals are so important in the RE process.

- Achieving requirements completeness is a major RE concern.. 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 proved to be achieved from the specification and the properties known about the domain considered [Yue87].
- Avoiding irrelevant requirements is another major RE concern. Goals provide a precise criterion for requirements *pertinence*; a requirement is pertinent with respect to a set of goals in the domain considered if its specification is used in the proof of one goal at least [Yue87].
- Explaining requirements to stakeholders is another important issue. Goals provide the rationale for requirements, in a way similar to design goals in design processes [Mos85, Lee91]. A requirement appears because of some underlying goal which provides a base for it [Ros77, Dar91, Som97]. More explicitly, a goal refinement tree provides traceability links from high-level strategic objectives to low-level technical requirements. In particular, for business application systems, goals may be used to relate the software-to-be to organizational and business contexts [Yu93].
- Goal refinement provides a natural mechanism for structuring complex requirements documents for increased readability. (This at least has been our experience in all industrial prjects we have been involved in, see Section 7.)
- Requirements engineers are faced with many alternatives to be considered during the requirements elaboration process. Our extensive experience revealed that alternative goal refinements provide the right level of abstraction at which decision makers can be involved for validating choices being made or suggesting other alternatives overlooked so far. Alternative goal refinements allow alternative system proposals to be explored [Lam00c].
- Managing conflicts among multiple viewpoints is another major RE concern [Nus94]. Goals have been recognized to provide the roots for detecting conflicts among requirements and for resolving them eventually [Rob89, Lam98b].
- Separating stable from more volatile information is another important concern for managing requirements evolution. A requirement represents one particular way of achieving some specific goal; the requirement is therefore more likely to evolve, towards another way of achieving that same goal, than the goal itself. The higher level a goal is, the more stable it will be. Others have made that same observation [Ant94]. It turns out that different system versions often share a common set of high-level goals; the current system and the system-to-be correspond to alternative refinements of common goals in the goal refinement graph, and can therefore be integrated into one single goal model (see Section 3).
- Last but not least, goals drive the identification of requirements to support them; they have been shown to be among the basic driving forces, together with scenarios, for a systematic requirements elaboration process [Dar91, Rub92, Dar93, Ant98, Dub98, Kai00, Lam00c]. We will come back to this in Sections 5 and 6.

Where are goals coming from?

Goal identification is not necessarily an easy task [Lam95, Ant98, Hau98, Rol98]. Sometimes they are explicitly stated by stakehokders or in preliminary material available to requirements engineers. Most often they are implicit so that goal elicitation has to be undertaken.

The preliminary analysis of the current system is an important source for goal identification. Such analysis usually results in a list of problems and deficiencies that can be for-

mulated precisely. Negating those formulations yields a first list of goals to be achieved by the system-to-be.

In our experience, goals can also be identified systematically by searching for intentional keywords in the preliminary documents provided, interview transcripts, etc. [Lam00c].

Once a preliminary set of goals and requirements is obtained and validated with stakeholders, many other goals can be identified by *refinement* and by *abstraction*, just by asking HOW and WHY questions about the goals/requirements already available, respectively [Lam95, Lam00c].

More sophisticated techniques for goal refinement and abstraction (notably, from scenarios) will be reviewed in Section 5. Other goals are identified by resolving conflicts among goals or obstacles to goal achievement, see Section 5 too.

A common misunderstanding about goal-oriented approaches is that they are inherently top-down; this is by no means the case as it should hopefully be clear now from the discussion above.

When should goals be made explicit?

It is generally argued that goal models are built during the early phases of the RE process [Dar93, Yu97, Dub98]. The basis for the argument is the driving role played by goals in that process; the soonest a goal is identified and validated, the best. This does not imply any sort of waterfall-like requirements elaboration process, however. As requirements "implement" goals much the same way as programs implement design specifications, there is some inevitable intertwining of goal identification and requirements elaboration [Lam95, Swa82]. Goals may thus sometimes be identified fairly lately in the RE process --especially when WHY questions about technical details or scenarios, initially taken for granted, are raised lately in the process.

3. Modeling goals

The benefit of goal modeling is to support heuristic, qualitative or formal reasoning schemes during requirements engineering (see Section 5). Goals are generally modelled by *intrinsic features* such as their type and attributes, and by their *links* to other goals and to other elements of a requirements model.

Goal types and taxonomies. Goals can be of different types. Several classification axes have been proposed in the literature.

Functional goals underlie services that the system is expected to deliver whereas *non-functional* goals refer to expected system qualities such as security, safety, performance, usability, flexibility, customizability, interoperability, and so forth [Kel90]. This typology is overly general and can be specialized. For example, *satisfaction* goals are functional goals concerned with satisfying agent requests; *information* goals are functional goals concerned with keeping such agents informed about object states [Dar93]. Non-functional goals can be specialized in a similar way. For example, *accuracy* goals are non-functional goals requiring the state of software objects to accurately reflect the state of the corresponding monitored/controlled objects in the environment [Myl92, Dar93] --such goals are often overlooked in the RE process; their violation may be responsible for major failures [Lam00a]. *Performance* goals are specialized into time and space performance goals, the former being specialized into *response time* and *throughput* goals [Nix93]. *Security* goals are specialized into *confidentiality*, *integrity* and *availability* goals [Amo94]; the latter can be specialized in turn until reaching domain-specific security goals. A rich taxonomy for non-functional goals can be found in [Chu00].

Another distinction often made in the literature is between *soft goals*, whose satisfaction cannot be established in a clear-cut sense [Myl92], and (hard) goals whose satisfaction can be established through verification techniques [Dar93, Dar96]. Soft goals are especially useful for comparing alternative goal refinements and chosing one that contributes the "best" to them, see below.

Another classification axis is based on types of temporal behaviour prescribed by the goal. [Dar93]. *Achieve* (resp. *cease*) goals generate system behaviours, in that they require some target property to be eventually satisfied in some future state (resp. denied); *maintain* (resp. *avoid*) goals retrict behaviours, in that they require some target property to be permanently satisfied in every future state (resp. denied) unless some other property holds. *Optimize* goals compare behaviours to favor those which better ensure some soft target property.

In a similar vein, [Sut93] proposes a classification according to desired system states (e.g., positive, negative, alternative, feedback, or exception-repair) and to goal level (e.g., policy level, functional level, domain level). [Ant94] makes a distinction beween objective goals, that refer to objects in the system, and adverbial goals, that refer to ways of achieving objective goals.

Goal types and taxonomies are used to define heuristics for goal acquisition, goal refinement, requirements derivation, and semi-formal consistency/completeness checking [Dar93, Sut93, Ant98, Chu00, Ant01], or to retrieve goal specifications in the context of specification reuse [Mas97].

Goal attributes. Beside their type, goals can also be intrinsically characterized by attributes such as their *name* and their *specification* (see Section 4). *Priority* is another important attribute that can be attached to goals [Dar93]. Qualitative values for this attribute allow mandatory or optional goals to be modelled with various degrees of optionality. Priorities are often used for resolving conflicts among goals [Rob89, Lam98b]. Other goal attributes that have been proposed include goal *utility* and *feasibility* [Rob89].

Goal Links. Many different types of links have been introduced in the literature to relate goals (a) with each other and (b) with other elements of requirements models. Such links form the basis for defining *goal structures*. We discuss inter-goal links first, and then links between goals and other elements of requirements models such as agents, scenarios, or operations.

Links between goals are aimed at capturing situations where goals positively or negatively *support* other goals. Directly borrowed from problem reduction methods in Artificial

Intelligence [Nil71], AND/OR graphs may be used to capture goal *refinement* links [Dar91, Dar93]. *AND-refinement* links relate a goal to a set of subgoals (called refinement); this means that satisfying all subgoals in the refinement is sufficient for satisfying the parent goal. *OR-refinement* links relate a goal to an alternative set of refinements; this means that satisfying one of the refinements is sufficient for satisfying the parent goal. In this framework, a *conflict* link between two goals is introduced when the satisfaction of one of them may prevent the other from being satisfied. Those link types are used to capture alternative goal refinements and potential conflicts, and to prove the correctness of goal refinements (see Section 5).

Weaker versions of those link types have been introduced to relate soft goals [Rob89, Myl92, Chu00] as the latter can rarely be said to be satisfied in a clear-cut sense. Instead of goal satisfaction, goal *satisficing* is introduced to express that subgoals are expected to achieve the parent goal within acceptable limits, rather than absolutely. A subgoal is then said to *contribute* partially to the parent goal, regardless of other subgoals; it may contribute *positively* or *negatively.* The semantic rules are now as follows. If a goal is AND-decomposed into subgoals and all subgoals are satisficed, then the parent goal is satisficeable; but if a subgoal is denied then the parent goal is deniable. If a goal contributes negatively to another goal and the former is satisficed, then the latter is deniable. These rules are used for qualitative reasoning about goal satisficing (see Section 5).

Beside inter-goal links, goals are in general also linked to other elements of requirements models. KAOS introduces AND/OR *operationalization* links to relate goals to the operations which ensure them through corresponding required pre-, post-, and trigger conditions [Lam98c, Lam00c] (the older notion of operationalization [Dar91, Dar93] was revised and simplified from practical experience). Others have used similar links between goals and operations, e.g., [Ant94, Ant98, Kai00]. In [Myl92], the inter-goal *contribution* link types are extended to capture the positive/negative contribution of requirements to goals; *argumentation* links are also introduced to connect supporting arguments to contribution links.

There has been a massive amount of work on linking goals and scenarios together --e.g., [Fic92, Dar93, Pot95, Lei97, Sut98, Ant98, Hau98, Lam98b, Rol98, Kai00, Ant01]. The obvious reason is that scenarios and goals have complementary characteristics; the former are concrete, narrative, procedural, and leave intended properties implicit; the latter are abstract, declarative, and make intended properties explicit. Scenarios and goals thus complement each other nicely for requirements elicitation and validation. By and large the link between a goal and a scenario is a *coverage link*; the main differences between the various modeling proposals lie in the fact that a scenario may be type-level or instance-level, may be an example or a counter-example of desired behavior, and may exercise a goal or an obtsacle to goal achievement.

Goal models may also be related to object models as goal formulations refer to specific objects, e.g., entities, relationships or agents [Dar93]. This link type allows pertinent object models to be systematically derived from goal models [Lam00c].

Various proposals have also been made to relate goals to agents. In KAOS, *responsibility* links are introduced to relate the goal and agent submodels. A goal may be assigned to alternative agents through OR responsibility links; this allows alternative boundaries to be explored between the software-to-be and its environment. “Responsibility” means that the agent is committed to restrict its behavior by performing the operations it is assigned to only under restricted conditions, namely, those prescribed by the required pre-, post-, and trigger conditions [Dar93]. This notion of responsibility derives from [Fea87, Fin87]; it is studied in depth in [Let01]. *Wish* links are also sometimes used in heuristics for agent assignment [Dar91]; e.g., one should avoid assigning a goal to an agent wishing other goals in conflict with that goal..

In the i* framework [Yu93, Yu97], various types of agent dependency links are defined to model situations where an agent depends on another for a goal to be achieved, a task to be achieved, or a resource to become available. For each type of dependency an operator is defined; operators may be combined to define plans that agents may use to achieve goals. The purpose of this modelling is to support various kinds of checks such as the viability of an agent's plan or the fulfilment of a commitment between agents. Although initially conceived for modeling the organizational environment of the software-to-be, the TROPOS project is currently aiming at propagating this framework to later stages of the software lifecycle, notably, for modeling agent-oriented software architectures.

Various authors have also suggested representing the links between goals and organizational policies, e.g., [Sib93, Fea93, Sut93].

At the process level, it may be useful for traceability purpose [Got95] to record which actor owns which goal or some view of it [Lam98b].

4. Specifying goals

Goals must obviously be specified precisely to support requirements elaboration, verification/validation, conflict management, negotiation, explanation and evolution.

An informal (but precise) specification should always be given to make it precise what the goal name designates [Zav97a].

Semi-formal specifications generally *declare* goals in terms of their type, attribute, and links (see Section 3). Such declarations may in general be provided alternatively using a textual or a graphical syntax (see, e.g., [Dar98]). In the NFR framework [Myl92], a goal is specified by the most specific subtype it is an instance of, parameters that denote the object attributes it refers to, and the degree of satisficing/denial by child goals. Semi-formal specifications often include keyword verbs with some predefined semantics. For example, Achieve, Maintain and Avoid verbs in KAOS specify a temporal logic pattern for the goal name appearing as parameter [Dar93]; they implicitly specify that a corresponding target condition should hold some time in the future, always in the

future unless some other condition holds, or never in the future. The intent is to provide a lightweight alternative to full formalization of the goal formulation, still amenable to some form of analysis. This basic set has been extended with qualitative verbs such as Improve, Increase, Reduce, Make, and so forth [Ant98]. In a similar spirit, goals in [Rol98] are represented by verbs with different parameters playing different roles with respect to the verb --e.g., target entities affected by the goal, beneficiary agents of the goal achievement, resource entities needed for goal achievement, source or destination of a communication goal, etc.

Formal specifications *assert* the goal formulation in a fully formal system amenable to analysis. In KAOS, such assertions are written in a real-time linear temporal logic heavily inspired from [Man92, Koy92] with the usual operators over past and future states, bound by time variables; semantically, they capture maximal sets of desired behaviors [Dar93, Let01]. The KAOS language is "2-button" in that the formal assertion layer is optional; it is used typically for critical aspects of the system only.

More formal specifications yield more powerful reasoning schemes at the price of higher specification effort and lower usability by non-experts; the various techniques briefly reviewed here should thus be seen as complementary means rather than alternative ones; their suitability may heavily depend on the specific type of system being considered.

5. Reasoning about goals

The ultimate purpose of goal modelling and specification is to suport some form of goal-based reasoning for RE subprocesses such as requirements elaboration, consistency and completeness checking, alternative selection, evolution management, and so forth.

5.1 Goal verification

One of the benefits of goal-oriented RE is that one can verify that the requirements entail the goals identified, and check that the set of requirements specified is sufficiently complete for the set of goals identified [Yue87]. More precisely, if *R* denotes the set of requirements, *As* the set of environmental assumptions, *D* the set of domain properties, and *G* the set of goals, the following satisfaction relation must hold for each goal *g* in *G*::

$$R, As, D \models g \quad \text{with} \quad R, As, D \not\models \textbf{false}$$

This may be checked informally, or formally if the goal specifications and domain properties are formalized. For temporal logic specifications one may rely on the proof theory of temporal logic and use tools such as, e.g., STeP [Man96].

A lightweight alternative is to use formal refinement patterns fo Achieve, Maintain and Avoid goals [Dar96]. Such patterns are proved correct and complete once for all; refinements in the goal graph are then verified by matching them to one applicable pattern from the library. The mathematical proof intricacies are thereby hidden. A frequently used pattern is the decomposition-by-milestone pattern that refines a parent Achieve goal

$$P \Rightarrow \Diamond Q$$

into two subgoals:

$$P \Rightarrow \Diamond \mathbf{R} \,,\, \mathbf{R} \Rightarrow \Diamond Q$$

where the "◊" temporal operator means "sometime in the future". Another frequently used pattern is the decomposition-by-case pattern that refines the same parent Achieve goal into three subgoals:

$$P \wedge \mathbf{R} \Rightarrow \Diamond Q \,,\, P \Rightarrow \Diamond R \,,\, P \Rightarrow P \; W \; Q$$

where the "*W*" temporal operator means "always in the future unless".

The techniques above can be used for goals that can be said to be established in a clear-cut sense. For soft goals, the qualitative reasoning procedure provided by the NFR framework is particularly appropriate [Myl92]. This procedure determines the degree to which a goal is satisficed/denied by lower-level goals/requirements. A node or link in the goal graph is labelled *S* (satisficed) if it is satisficeable and not deniable; *D* (denied) if it is deniable but not satisficeable; *C* (conflicting) if it is both satisficeable and deniable; and *U* (undetermined) if it is neither satisficeable nor deniable. The general idea is to propagate such labels along satisficed links bottom-up, from lower-level nodes (i.e. requirements) to higher-level nodes (i.e. goals). Additional label values can be assigned at intermediate stages of the procedure, namely, U^+ (inconclusive positive support), U^- inconclusive negative support, and *?* (requiring user intervention to specify an appropriate label value). Rules for bottom-up propagation of labels are then defined accordingly. A example of application of this framework to performance goals can be found in [Nix93].

5.2 Goal validation

Goals can be validated by identifying or generating scenarios that are covered by them [Hau98]. One may even think of enacting such scenarios to produce animations [Hey98]. The scenario identification process is generally based on heuristics [Sut98, Ant98].

In [And89], plan-based techniques are used to tentatively generate scenarios showing that a goal can be achieved without reaching prohibited conditions. Goals, prohibited conditions and operations are specified formally by simple state predicates. An automated planner first produces a trial scenario to achieve the goal condition; it then checks for faults in the proposed scenario by looking for scenarios achieving the prohibited conditions; finally it assists the specifier in modifying the set of operations in case faults are found. [Fic92] explores this deficiency-driven paradigm further. The system is specified by a set of goals, formalized in some restricted temporal logic, a set of scenarios, expressed in a Petri net-like language, and a set of agents producing restricted scenarios to achieve the goals they are assigned to. The general approach consists of (a) trying to detect inconsistencies between scenarios and goals, and (b) applying operators that modify the specification to remove the inconsistencies. Step (a) is carried out by a planner that searches for behaviours leading to some goal violation. The operators offered to the analyst in Step (b) encode heuristics for specification debugging --e.g., introduce an agent whose responsibility is to prevent the state transitions that are the last step

in breaking the goal. There are operators for introducing new types of agents with appropriate responsibilities, splitting existing types, introducing communication and synchronization protocols between agents, weakening idealized goals, etc. The repeated application of deficiency detection and debugging operators allows the analyst to explore the design space and hopefully converge towards a satisfactory specification.

5.3 Goal-based requirements elaboration

The technique just sketched above is a first step towards making verification/validation contribute to the requirements elaboration process. The main reason for goal-oriented RE after all is to let goals help elaborating the requirements supporting them. A goal-based elaboration typically consists of a hybrid of top-down and bottom-up processes, plus additional processes driven by the handling of possible abnormal agent behaviors, the management of conflicting goals, the recognition of analogical situations from which specifications can be transposed, and so forth. Note, however, that for explanatory purpose the resulting requirements document is in general better presented in a top-down way.

Goal/requirement elicitation by refinement

An obvious (but effective) informal technique for finding out subgoals and requirements is to keep asking HOW questions about the goals already identified [Lam95, Lam00c].

Formal goal refinement patterns may also prove effective when goal specifications are formalized; typically, they help finding out subgoals that were overlooked but are needed to establish the parent goal. Consider a simple train control system, for example, and the functional goal of train progress through consecutive blocks:

Goal Achieve [TrainProgress]
FormalDef ($\forall$ t:r Train, b: Block) [On (tr, b) $\Rightarrow \Diamond$ On (tr, b+1)]

A particular case that comes directly to mind is when block *b+1*'s signal is set to 'go'. Two subgoals coming naturally to mind are the following:

Goal Achieve [ProgressWhenGoSignal]
FormalDef $\forall$ tr: Train, b: Block
On (tr, b) $\curlywedge$ *Go[b+1]* $\Rightarrow \Diamond$ On (tr, b+1)

Goal Achieve [SignalSetToGo]
FormalDef $\forall$ tr: Train, b: Block
On (tr, b) $\Rightarrow \Diamond$ Go[b+1]

This tentative refinement matches the decomposition-by-case pattern in Section 5.1 and therefore allows the following missing subgoal to be pointed out:

Goal Maintain [TrainWaiting]
FormalDef $\forall$ tr: Train, b: Block
On (tr, b) $\Rightarrow$ On (tr, b) *W* On (tr, b+1)

Another effective way of driving the refinement process is based on the determination that an agent candidate to goal assignment cannot realize the goal, e.g., because it cannot monitor the variables appearing in the goal antecedent or control the variables appearing in the goal consequent. [Let01] gives a set of conditions for goal unrealizability; this set is proved complete and provides the basis for a rich, systematic set of agent-driven refinements tactics for generating realizable subgoals and auxiliary agents.

Goal/requirement elicitation by abstraction

An obvious (but effective) informal technique for finding out more abstract, parent goals is to keep asking WHY questions about operational descriptions already available [Lam95, Lam00c].

More sophisticated techniques have been devised to elicit goals from scenarios. Based on a bidirectional coupling between type-level scenarios and goal verb templates as discussed in Section 4, [Rol98] proposes heuristic rules for finding out alternative goals covering a scenario (corresponding to alternative values for the verb parameters), missing companion goals, or subgoals of the goal under consideration. On a more formal side, [Lam98c] describes an inductive learning technique that takes scenarios as examples and counterexamples of intended behavior and generates goal specifications in temporal logic that cover all the positive scenarios and exclude all the negative ones.

Note also that refinement patterns when applied in the reverse way correspond to abstraction patterns that may produce more coarse-grained goals.

Goal operationalization

A few efforts have been made to support the process of deriving pre-, post-, and trigger conditions on software operations so as to ensure the terminal goals in the refinement process. The principle is to apply derivation rules whose premise match the goal under consideration [Dar93, Let01]. Consider, for example, the following goal:

Goal Maintain [DoorsClosedWhileMoving]
FormalDef $\forall$ tr: Train, loc, loc': Location
At (tr, loc) $\wedge$ o At (tr, loc') $\wedge$ loc <> loc'
$\Rightarrow$ tr.Doors = 'closed' $\wedge$ o (tr.Doors = 'closed')

where the "o" temporal operator means "in the next state". Applying the following derivation rule

$$G\text{: } P \wedge (P1 \wedge o\ P2 \Rightarrow Q1 \wedge o\ Q2)\ ,\ \textit{DomPre:}\ P1,\ \textit{DomPost:}\ P2$$
$$\text{---}$$
$$\textit{ReqPre}\ \text{for}\ G\text{: } Q1\ ,\ \textit{ReqPost}\ \text{for}\ G\text{: } Q2$$

we derive the following operationalization:

Operation Move
Input tr: Train; loc, loc': Location ; **Output** At
DomPre At (tr, loc) $\wedge$ loc <> loc'
DomPost At (tr, loc')
RequiredPre for DoorsClosedWhileMoving: *tr.Doors = 'closed'*
RequiredPost for DoorsClosedWhileMoving: *tr.Doors = 'closed'*

Analogical reuse

Goal-based specifications can also be acquired by retrieving structurally and semantically analog specifications in a repository of reusable specification components, and then transposing the specifications found according to the structural and semantic matching revealed by the retrieval process [Mas97].

Obstacle-driven elaboration

First-sketch specifications of goals, requirements and assumptions are often too ideal; they are likely to be violated from time to time in the running system due to unexpected behaviors of agents. The lack of anticipation of exceptional behaviors may result in unrealistic, unachievable and/or incomplete requirements.

Such exceptional behaviors are captured by assertions called *obstacles* to goal satisfaction. An obstacle *O* is said to obstruct a goal *G* in a domain *Dom* iff

$\{O, Dom\} \models \neg G$ *obstruction*

$Dom \models\!\!\!\not\models \neg O$ *domain consistency*

Obstacles thus need to be identified and resolved at RE time in order to produce robust requirements and hence more reliable software. The notion of obstacle was just mentioned in [Yue87]. It was elaborated further in [Pot95] where scenarios are shown to be a good vehicle for identifying goal obstructions. Some heuristics for identifying obstacles can be found in [Pot95] and [Ant98]. More formal techniques are described in [Lam98a] and then [Lam00a] for:

- the abductive generation of obstacles from goal specifications and domain properties,
- the systematic generation of various types of obstacle resolution, e.g., goal substitution, agent substitution, goal weakening, goal restoration, obstacle mitigation, or obstacle prevention.

Obstacles can also be resolved at run time in some cases, see [Fea98].

5.4 Conflict management

Requirements engineers live in a world where conflicts are the rule, not the exception [Eas94]. Conflicts generally arise from multiple viewpoints and concerns [Nus94]. They must be detected and eventually resolved even though they may be temporarily useful for eliciting further information [Hun98]. Various forms of conflict are studied in [Lam88b], in particular, a weak form called *divergence* which occurs frequently in practice.

The goals $G_1, ..., G_n$ are said to be *divergent* iff there exists a non-trivial *boundary condition B* such that :

$\{ B, \forall_i G_i, Dom\} \models$ **false** *inconsistency*

$\{ B, \forall_{j \neq i} G_j, Dom\} \models\!\!\!\not\models$ **false** *minimality*

("Non-trivial" means that *B* is different from the bottom **false** and the complement $\neg \forall_i G_i$). Note that the traditional case of conflict, in the sense of logical inconsistency, amounts to a particular case of divergence.

Divergences need to be identified and resolved at RE time in order to eventually produce consistent requirements. Formal and heuristic techniques are described in [Lam98b] for:

- the abductive generation of boundary conditions from goal specifications and domain properties,
- the systematic generation of various types of divergence resolution.

A qualitative procedure is suggested in [Rob89] for handling conflicts. The idea is to detect them at requirements level and characterize them as differences at goal level. The user of the procedure first identifies the requirements elements that correspond to each other in the various viewpoints at hand; conflict detection is then carried out by mapping syntactic differences between the corresponding requirements elements to differences in values of variables involved in the goals supported by these elements. Conflict resolution is attempted next by appealing to compromises (e.g., through compensations or restriction specialization), or goal substitutions. Finally, the conflict resolution at goal level is down propagated to the requirements level.

5.5 Goal-based negotiation

Conflict resolution often requires negotiation. [Boe95] proposes an iterative 3-step process model for goal-based negotiation of requirements. At each iteration of a spiral model for requirements elaboration,

(1) all stakeholders involved are identified together with their wished goals (called *win conditions*);

(2) conflicts between these goals are captured together with their associated risks and uncertainties;

(3) goals are reconciled through negotiation to reach a mutually agreed set of goals, constraints, and alternatives for the next iteration.

5.6 Alternative selection

Which goal refinement should be selected when alternative ones are identified? Which agent assignment should be selected when alternative ones are identified? This is by and large an open problem. There are local tactics of course, such as favoring alternatives with less critical obstacles or conflicts, but a systematic approach has not emerged so far in the RE literature.

One promising direction would be to use qualitative reasoning schhemes à la NFR [Myl92] to select an alternative refinement that contributes the best to the satisficing of soft goals related to cost, reliability, performance etc. Multicriteria analysis techniques could be helpful here.

6. A goal-oriented RE method in action

It is now time to demonstrate how some of the techniques reviewed above can fit together in a goal-oriented RE method. We come back to a case study we have already presented in [Lam00c] because it illustrates many of the issues raised here; the initial document is unbiased as it comes from an independent source involved in the development,; it is publically available [BAR99] --unlike most documents from the industrial projects we have been involved in; the system is a real, complex, real-time, safety-critical one (this allows one to suggest that goal-oriented RE is not only useful for business applications). The initial document focuses on the control of speed and acceleration of trains under responsibility of the Advanced Automatic Train Control being developed for the San Francisco Bay Area Rapid Transit (BART) system.

We follow the KAOS method [Dar93, Lam95, Lam00c] in order to incrementally elaborate four complementary submodels: (1) the goal model, (2) the object model; (3) the agent responsibility model, leading to alternative system boundaries; (4) the operation model. The goal refinement graph is elaborated by eliciting goals from available sources and asking *why* and *how* questions (*goal elaboration step*); objects, relationships and attributes are derived from the goal specifications (*object modeling step*); agents are identified, alternative responsibility assignments are explored, and agent interfaces are derived (*responsibility assignment step*);

operations and their domain pre- and postconditions are identified from the goal specifications, and strengthened pre-/postconditions and trigger conditions are derived so as to ensure the corresponding goals (*operationalization step*). These steps are not strictly sequential as progress in one step may prompt parallel progress in the next one or backtracking to a previous one.

The presentation will be sketchy for lack of space; the interested reader may refer to [Let01] for a much greater level of details.

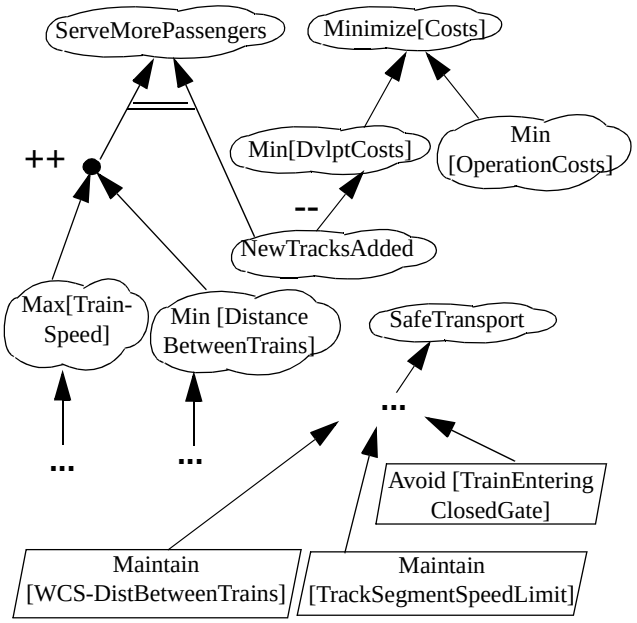

Figure 1 - Preliminary goal graph for the BART system

Goal identification from the initial document

A first set of goals is identified from a first reading of the available source [BART99] by searching for intentional keywords such as "objective", "purpose", "intent", "concern", "in order to", etc. A number of soft goals are thereby identified, e.g., "ServeMorePassengers", "NewTracksAdded", "Minimize[DevelopmentCosts]", "Minimize[DistanceBetweenTrains]", "SafeTransportation", etc. These goals are qualitatively related to each other through support links: Contributes (+), ContributesStrongly (++), Conflicts (-), ConflictsStrongly (- -). These weights are used to select among alternatives. Where possible, keywords from the semi-formal layer of the KAOS language are used to indicate the goal category. The *Maintain* and *Avoid* keywords specify "always" goals having the temporal pattern $\Box(P \rightarrow Q)$ and $\Box(P \rightarrow \neg Q)$, respectively. The *Achieve* keyword specifies "eventually" goals having the pattern $P \Rightarrow \Diamond Q$. The "$\rightarrow$" connective denotes logical implication; $\Box(P \rightarrow Q)$ is denoted by $P \Rightarrow Q$ for short.

Figure 1 shows the result of this first elicitation. Clouds denote soft-goals, parallelograms denote formalizable goals, arrows denote goal-subgoal links, and a double line linking arrows denotes an OR-refinement into alternative subgoals.

Formalizing goals and identifying objects

The object modeling step can start as soon as goals can be formulated precisely enough. The principle here is to identify objects, relationships and attributes from goal specifications. Consider, for example, the following goal at the bottom of Figure 1:

Goal Maintain[TrackSegmentSpeedLimit]
InformalDef *A train should stay below the maximum speed the track segment can handle.*
FormalDef $\forall$ tr: Train, s: TrackSegment :
On(tr, s) $\Rightarrow$ tr.Speed $\leq$ s.SpeedLimit

From the predicate, objects, and attributes appearing in this goal formalization we derive the following portion of the object model:

Train	On	TrackSegment
Speed: *SpeedUnit*		SpeedLimit: *SpeedUnit*
...		...

Similarly, the other goal at the bottom of Figure 5 is specified as follows:

Goal Maintain[WCS-DistBetweenTrains]
InformalDef *A train should never get so close to a train in front so that if the train in front stops suddenly (e.g., derailment) the next train would hit it.*
FormalDef $\forall$ tr1, tr2: Train :
Following(tr1, tr2) $\Rightarrow$ tr1.Loc - tr2.Loc > tr1.WCS-Dist

(The InformalDef statements in those goal definitions are taken literally from the initial document; *WCS-Dist* denotes the physical worst-case stopping distance based on the physical speed of the train.) This new goal specification allows the above portion of the object model to be enriched with *Loc* and *WCS-Dist* attributes for the *Train* object together with a reflexive *Following* relationship on it. The formalization of the goal Avoid[TrainEnterinClosedGate] in Figure 1 will further enrich the object model by elements that are strictly necessary to the goals considered. *Goals thus provide a precise driving criterion for identifying elements of the object model.*

Eliciting new goals through WHY questions

It is often the case that higher-level goals underpinning goals easily identified from initial sources are kept implicit in such sources. They may, however, be useful for finding out other important subgoals of the higher-level goal that were missing for the higher-level goal to be achieved.

As mentioned before, higher-level goals are identified by asking WHY questions about the goals available.

For example, asking a WHY question about the goal Maintain[WCS-DistBetweenTrains] yields the parent goal Avoid[TrainCollision]; asking a WHY question about the goal Avoid[TrainEnteringClosedGate] yields a new portion of the goal graph, shown in Figure 2.

In this goal subgraph, the companion subgoal Maintain[GateClosedWhenSwitchInWrongPosition] was elicited *formally* by matching a formal refinement pattern to the formalization of the parent goal Avoid[TrainOnSwitchInWrongPosition], found by a WHY question, and to the formalization of the initial goal Avoid[TrainEnteringClosedGate] [Dar96, Let01]. The dot joining the two lower refinement links together in Figure 2

means that the refinement is (provably) complete.

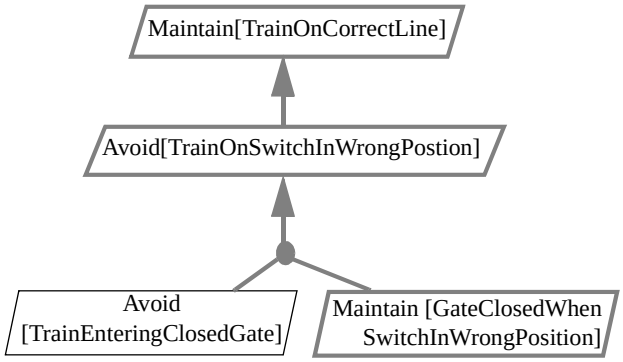

Figure 2 - Enriching the goal graph by WHY elicitation

Eliciting new goals through HOW questions

Goals need to be refined until subgoals are reached that can be assigned to individual agents in the software-to-be and in the environment. Terminal goals become requirements in the former case and assumptions in the latter.

More concrete goals are identified by asking HOW questions. For example, a HOW question about the goal Maintain[WCS-DistBetweenTrains] in Figure 1 yields an extension of the goal graph shown in Figure 3.

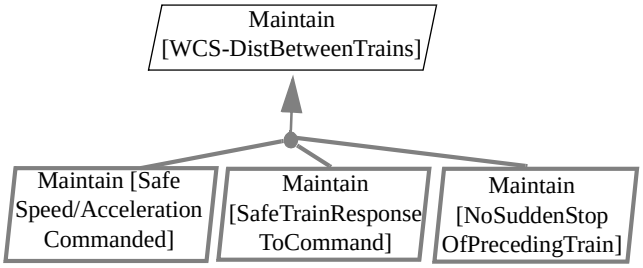

Figure 3 - Goal refinement

The formalization of the three subgoals in Figure 3 may be used to prove that together they entail the parent goal Maintain[WCS-DistBetweenTrains] formalized before [Let01]. These subgoals need be refined in turn until assignable subgoals are reached. A complete refinement tree is given in Annex 1.

Identifying potential responsibility assignments

Annex 1 also provides a possible goal assignment among individual agents. This assignment seems the one suggested in the initial document [BAR99]. For example, the accuracy goal Maintain[AccurateSpeed/PositionEstimates] is assignable to the TrackingSystem agent; the goal Maintain[SafeTrainResponse-ToCommand] is assignable to the OnBoardTrainController agent; the goal Maintain[SafeCmdMsg] is assignable to the Speed/AccelerationControlSystem agent.

It is worth noticing that goal refinements and agent assignments are both captured by AND/OR relationships. Alternative refinements and assignments can be (and probably have been) explored. For example, the parent goal Maintain[WCS-DistBetweenTrains] in Figure 3 may alternatively be refined by the following three Maintain subgoals:

PreceedingTrainSpeed/PositionKnownToFollowingTrain
SafeAccelerationBasedOnPreceedingTrainSpeed/Position
NoSuddenStopOfPreceedingTrain

The second subgoal above could be assigned to the OnBoard-TrainController agent. This alternative would give rise to a fully distributed system.

As suggested before, qualitative reasoning techniques in the style of [Myl99] might be applied to the softgoals identified in Figure 1 to help making choices among alternatives.

Deriving agent interfaces

Let us now assume that the goal Maintain[SafeCmdMsg] at the bottom of the tree in Annex 1 has been actually assigned to the Speed/AccelerationControlSystem agent. The interfaces of this agent in terms of monitored and controlled variables can be derived from the formal specification of this goal (we just take its general form here for sake of simplicity):

Goal Maintain[SafeCmdMsg]
FormalDef ∀ cm: *CommandMessage*, ti1, ti2: *TrainInfo*
cm.Sent ∧ cm.TrainID = ti1.TrainID ∧ FollowingInfo (ti1, ti2)
⇒ cm.Accel ≤ F (ti1, ti2) ∧ cm.Speed > G (ti1)

To fulfil its responsibility for this goal the Speed/Acceleration-ControlSystem agent must be able to *evaluate* the goal antecedent and *establish* the goal consequent. The agent's monitored variable is therefore *Train.Info* whereas its controlled variables are *CommandMessage.Accel* and *CommandMessage.Speed*. The latter will in turn become monitored variables of the OnBoardTrainController agent, by similar analysis. The technique for deriving the agent's monitored and controlled variables is fairly systematic, see [Let01] for details.

Identifying operations

The operationalization step starts by identifying the operations relevant to goals and defining their domain pre- and postconditions. Goals refer to specific state transitions; for each such transition an operation causing it is identified; its domain pre- and postcondition capture the state transition. For the goal Maintain[SafeCmdMsg] formalized above we get, for example,

Operation SendCommandMessage
Input Train {**arg** tr}
Output ComandMessage {**res** cm}
DomPre ¬ cm.Sent
DomPost cm.Sent ∧ cm.TrainID = tr.ID

This definition minimally captures what any sending of a command to a train is about in the domain considered; it does not ensure any of the goals it should contribute to.

Operationalizing goals

The next operationalization sub-step is to strengthen such domain conditions so that the various goals linked to the operation are ensured. For goals assigned to software agents, this step produces *requirements* on the operations for the corresponding goals to be achieved. As mentioned before, derivation rules for an operationalization calculus are available [Dar93, Let01]. In our example, they yield the following requirements that strengthen the domain pre- and postconditions:

Operation SendCommandMessage
Input Train {**arg** tr}, TrainInfo; **Output** ComandMsg {**res** cm}
DomPre ... ; **DomPost** ...
ReqPost for SafeCmdMsg:
Tracking (ti1, tr) ∧ Following (ti1, ti2)
→ cm.Acc ≤ F (ti1, ti2) ∧ cm.Speed > G (ti1)
ReqTrig for CmdMsgSentInTime:
$\blacksquare_{\leq 0.5\ sec}$ ¬ ∃ cm2: CommandMessage:
cm2.Sent ∧ cm2.TrainID = tr.ID

(The trigger condition captures an obligation to trigger the operation as soon as the condition gets true and provided the domain precondition is true. In the example above the condition says that no command has been sent in every past state up to one half-second [BAR99].)

Using a mix of semi-formal and formal techniques for goal-oriented requirements elaboration, we have reached the level at which most formal specification techniques would start.

Anticipating obstacles

As mentioned before, goals also provide a basis for early generation of high-level exceptions which, if handled properly at requirements engineering time, may generate new requirements for more robust systems.

The following obstacles were generated to obstruct the subgoal Achieve[CommandMsgIssuedInTime]:

CommandMsgNotIssued,
CommandMsgIssuedLate,
CommandMsgSentToWrongTrain

For the companion subgoal Achieve[CommandMsgDeliveredInTime] we similarly generated obstacles such as:

CommandMsgDeliveredLate,
CommandMsgCorrupted

The last companion subgoal Maintain[SafeCmdMsg] may be obstructed by the condition

UnsafeAcceleration,

and so on. The obstacle generation process for a single goal results in a goal-anchored fault-tree, that is, a refinement tree whose root is the goal negation. Compared with standard fault-tree analysis [Lev95], obstacle analysis is goal-oriented, formal, and produces obstacle trees that are provably complete with respect to what is known about the domain [Lam00a].

Alternative obstacle resolutions may then be generated to produce new or alternative requirements. For example, the obstacle CommandMsgSentLate above could be resolved by an alternative design in which accelerations are calculated by the on-board train controller instead; this would correspond to a *goal substitution* strategy. The obstacle UnsafeAcceleration above could be resolved by assigning the responsibility for the subgoal SafeAccelerationCommanded of the goal Maintain[SafeCmdMsg] to the VitalStationComputer agent instead [BART99]; this would correspond to an *agent substitution* strategy. An *obstacle mitigation* strategy could be applied to resolve the obstacle OutOfDateTrainInfo obstructing the accuracy goal Maintain[AccurateSpeed/PositionEstimates], by introducing a new subgoal of the goal Avoid[TrainCollisions], namely, the goal Avoid[CollisionWhenOutOfDateTrainInfo]. This new goal has to be refined in turn, e.g., by subgoals requiring full braking when the message origination time tag has expired.

Handling conflicts

The initial BART document suggests an interesting example of divergence [BART99, p.13]. Roughly speaking, the train commanded speed may not be too high, because otherwise it forces the distance between trains to be too high, in order to achieve the DistanceIncreasedWithCommandedSpeed subgoal of the SafeTransportation goal; on the other hand, the commanded speed may not be too low, in order to achieve the LimitedAccelerAbove7mphOfPhysicalSpeed subgoal of the SmoothMove goal. There seems to be a flavor of divergence here.

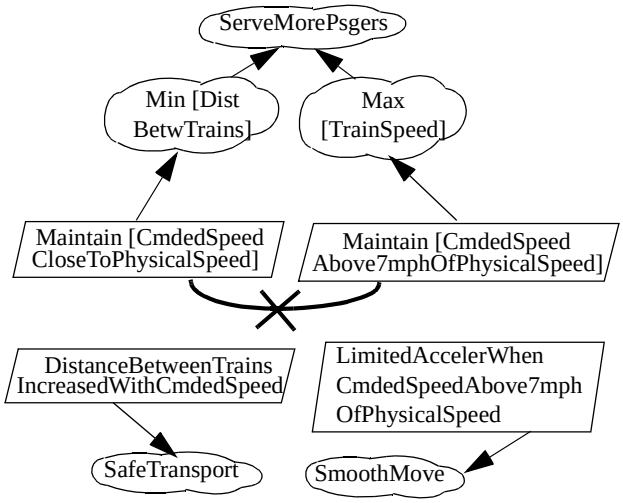

Figure 4 - Conflict in speed/acceleration control

We therefore look at the formalization of the suspect goals:

Goal Maintain [CmdedSpeedCloseToPhysicalSpeed]
FormalDef ∀ tr: Train
$tr.Acc_{CM} \geq 0$
$\Rightarrow tr.Speed_{CM} \leq tr.Speed + f(dist\text{-}to\text{-}obstacle)$

and

Goal Maintain [CmdedSpeedAbove7mphOfPhysicalSpeed]
FormalDef ∀ tr: Train
$tr.Acc_{CM} \geq 0 \Rightarrow tr.Speed_{CM} > tr.Speed + 7$

These two goals are formally detected to be divergent using the techniques described in [Lam98b]. The generated boundary condition for making them logically inconsistent is

$\Diamond$ (∃ tr: Train) ($tr.Acc_{CM} \geq 0 \wedge f(dist\text{-}to\text{-}obstacle) \leq 7$)

The resolution operators from [Lam98b] may be used to generate possible resolutions; in this case one should keep the safety goal as it is and weaken the other conflicting goal to remove the divergence:

Goal Maintain [CmdedSpeedAbove7mphOfPhysicalSpeed]
FormalDef ∀ tr: Train
$tr.Acc_{CM} \geq 0 \Rightarrow tr.Speed_{CM} > tr.Speed + 7$
$\vee$ *f (dist-to-obstacle)* ≤ *7*

7. Experience and tool support

The purrpose of this paper is obviously not to deliver an experience report. We would just like to mention here that

experience with goal-oriented requirements engineering is growing significantly, in different domain, different types of projects, and different project sizes. For example, Anton and colleagues have reported their experience with BPR applications [Ant94] and various electronic commerce systems [Ant98, Ant01]. Our understanding is that the NFR and i* frameworks have been experienced in real settings as well.

Our KAOS method has been used in 11 industrial projects to date. These include the goal-oriented reengineering of a complex, unintelligible requirements document for a phone system on TV cable; the goal-oriented modeling of a complex air traffic control application; the goal-oriented engineering of requirements for a variety of systems such as: a copyright management system for a major editor of cartoon strips, a management system for a hospital emergency service, a drug delivery management system for a big drug distributor, a new information system for a big daily newspaper, a web-based job information server, a web-based language translation system, and various e-learning systems. To give an idea, the copyright management system has 65 goals, 75 entity types and relationships, 11 agents, and 45 operations; the goal-oriented deliverable is 115 pages long. The size of the goal refinement graph for the other applications ranges from 50 to 100 goals and requirements.

Those projects could not have been undertaken without tool support. Our current GRAIL environment provides a graphical editor tightly coupled with a syntax-directed editor, an object-oriented specification database server supporting queries for model analysis, static semantics checkers, view filtering mechanisms, a HTML generator for model browsing in hypertext mode, and various types of report generators. Current efforts are devoted to an open, full Java version; the plan then is to integrate more formal support such as animators, model checkers, test data generators, formal verification tools, and so forth.

8. Goal orientation beyond RE

It has been suggested recently that the functional and (especially) non-functional goals elaborated in the RE process could be used for deriving and refining architectures [Lam00c] and for annotating design patterns [Chu00]. These are just preliminary efforts that should be expanded in a near future.

9. Conclusion

Goal-oriented requirements engineering has many advantages, some of which were recurrently felt in the aforementioned projects, to restate a few of them:

- object models and requirements can be derived systematically from goals;
- goals provide the rationale for requirements;
- a goal graph provides vertical traceability from high-level strategic concerns to low-level technical details; it allows evolving versions of the system under consideration to be integrated as alternatives into one single framework;
- goal AND/OR graphs provide the right abstraction level at which decision makers can be involved for important decisions;
- the goal refinement structure provides a comprehensible structure for the requirements document;
- alternative goal refinements and agent assignments allow alternative system proposals to be explored;
- goal formalization allows refinements to be proved correct and complete.

We hope to have convinced the reader that this area of RE is worth pursuing. There are many open issues to work on in the future, of course; the reader may refer to [Lam00c] for a discussion of them.

Acknowledgment. Discussions with Robert Darimont and Emmanuel Letier were a permanent source of inspiration and confrontation of some of the issues raised in this paper; they were in particular instrumental in developing KAOS specifications for various non-trivial systems, including the one outlined here [Let01]. I am also grateful to the KAOS/GRAIL crew at CEDITI for using some of the ideas presented here in industrial projects and providing regular feedback, among others, Emmanuelle Delor, Philippe Massonet, and André Rifaut. All the people whose work is mentioned in this paper had some influence on it in some way or another (whether they recognize and like it or not!).

References

[Amo94] E.J. Amoroso, Fundamentals of Computer Security. Prentice-Hall, 1994.

[And89] J.S. Anderson and S. Fickas, "A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem", *Proc. 5th Intl. Workshop on Software Specification and Design*, IEEE, 1989, 177-184.

[Ant94] A.I. Anton, W.M. McCracken, and C. Potts, "Goal Decomposition and Scenario Analysis in Business Process Reengineering, *Proc. CAISE'94*, LNCS 811, Springer-Verlag, 1994, 94-104.

[Ant98] A.I. Anton and C. Potts, "The Use of Goals to Surface Requirements for Evolving Systems", *Proc. ICSE-98: 20th Intrnational Conference on Software Enginering*, Kyoto, April 1998.

[Ant01] A.I. Anton, R. Carter, A. Dagnino, J. Dempster and D.F. Siege, "Deriving Goals from a Use-Case Based Requirements Specification", *Requirements Engineering Journal*, Vol. 6, 2001, 63-73.

[BAR99] Bay Area Rapid Transit District, Advance Automated Train Control System, Case Study Description. Sandia National Labs, http://www.hcecs.sandia.gov/bart.htm.

[Ber91] V. Berzins and Luqi, *Software Engineering with Abstractions*. Addison-Wesley, 1991.

[Boe95] B. W. Boehm, P. Bose, E. Horowitz, and Ming June Lee, "Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach", *Proc. ICSE-17 - 17th Intl. Conf. on Software Engineering*, Seattle, 1995, pp. 243-253.

[Chu00] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, *Non-functional requirements in software engineering*. Kluwer Academic, Boston, 2000.

[Dar91] A. Dardenne, S. Fickas and A. van Lamsweerde, "Goal-Directed Concept Acquisition in Requirements Elicitation", *Proc. IWSSD-6 - 6th Intl. Workshop on Software Specification and Design*, Como, 1991, 14-21.

[Dar93] A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, Vol. 20, 1993, 3-50.

[Dar96] R. Darimont and A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration", *Proc. FSE'4 - Fourth ACM SIGSOFT Symp. on the Foundations of Software Engineering*, San Francisco, October 1996, 179-190.

[Dar98]R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde, “GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering”, *Proc. ICSE’98 - 20th Intl. Conf. on Software Engineering*, Kyoto, April 1998, vol. 2, 58-62. (Earlier and shorter version found in *Proc. ICSE’97 - 19th Intl. Conf. on Software Engineering*, Boston, May 1997, 612-613.)

[Dub98] E. Dubois, E. Yu and M. Petit, "From Early to Late Formal Requirements: A Process-Control Case Study”, *Proc. IWSSD’98 - 9th International Workshop on Software Specification and Design*, Isobe, IEEE CS Press, April 1998, 34-42.

[Dwy99] M.B. Dwyer, G.S. Avrunin and J.C. Corbett, “Patterns in Property Specifications for Finite-State Verification”, *Proc. ICSE-99: 21th Intrnational Conference on Software Enginering*, Los Angeles, 411-420.

[Eas94] S. Easterbrook, “Resolving Requirements Conflicts with Computer-Supported Negotiation”. In *Requirements Engineering: Social and Technical Issues*, M. Jirotka and J. Goguen (Eds.), Academic Press, 1994, 41-65.

[Fea87] M. Feather, “Language Support for the Specification and Development of Composite Systems”, *ACM Trans. on Programming Languages and Systems* 9(2), Apr. 87, 198-234.

[Fea93] M. Feather, "Requirements Reconnoitering at the Juncture of Domain and Instance", *Proc. RE’93 - 1st Intl. IEEE Symp. on Requirements Engineering*, Jan. 1993, 73-77.

[Fea98] M. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard, “Reconciling System Requirements and Runtime Behaviour”, *Proc. IWSSD’98 - 9th International Workshop on Software Specification and Design*, Isobe, IEEE CS Press, April 1998.

[Fic92] S. Fickas and R. Helm, “Knowledge Representation and Reasoning in the Design of Composite Systems", *IEEE Trans. on Software Engineering*, June 1992, 470-482.

[Fin87] A. Finkelstein and C. Potts, "Building Formal Specifications Using Structured Common Sense", Proc. IWSSD-4 - 4th International Workshop on Software Specification and Design (Monterey, Ca.), IEEE, April 1987, 108-113.

[Fow97] M. Fowler, *UML Distilled*. Addison-Wesley, 1997.

[Got95] O. Gotel and A. Finkelstein, “Contribution Structures”, *Proc. RE’95 - 2nd Intl. IEEE Symp. on Requirements Engineering*, York, IEEE, 1995, 100-107.

[Gro01] D. Gross and E. Yu, “From Non-Functional Requirements to Design through Patterns”, *Requirements Engineering Journal* Vol. 6, 2001, 18-36.

[Hau98] P. Haumer, K. Pohl, and K. Weidenhaupt, “Requirements Elicitation and Validation with Real World Scenes”, *IEEE Trans. on Sofware. Engineering*, Special Issue on Scenario Management, December 1998, 1036-1054.

[Hey98] P. Heymans and E. Dubois, “Scenario-Based Techniques for Supporting the Elaboration and the Validation of Formal Requirements”, *Requirements Engineering Journal* Vol. 3 No. 3-4, 1998, 202-218.

[Hic74]G.F. Hice, W.S. Turner, and L.F. Cashwell, *System Development Methodology*. North Holland, 1974.

[Hun98] A. Hunter and B. Nuseibeh, “Managing Inconsistent Specifications: Reasoning, Analysis and Action”, *ACM Transactions on Software Engineering and Methodology*, Vol. 7 No. 4. October 1998, 335-367.

[Jac95] M. Jackson, Software Requirements & Specifications - A Lexicon of Practice, Principles and Pejudices. ACM Press, Addison-Wesley, 1995.

[Jar93] M. Jarke and K. Pohl, “Vision-Driven Requirements Engineering”, *Proc. IFIP WG8.1 Working Conference on Information System Development Process*, North Holland, 1993, 3-22.

[Kai00] H. Kaindl, “A Design Process Based on a Model Combining Scenarios with Goals and Functions”, *IEEE Trans. on Systems, Man and Cybernetic*, Vol. 30 No. 5, September 2000, 537-551.

[Kel90] S.E. Keller, L.G. Kahn and R.B. Panara, "Specifying Software Quality Requirements with Metrics", in Tutorial: System and Software Requirements Enginering, R.H. Thayer and M. Dorfman, Eds., IEEE Computer Society Press, 1990, 145-163.

[Koy92] R. Koymans, *Specifying message passing and time-critical systems with temporal logic*, LNCS 651, Springer-Verlag, 1992.

[Lam95] A. van Lamsweerde, R. Darimont, and Ph. Massonet, "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt", *Proc. RE’95 - 2nd Intl. IEEE Symp. on Requirements Engineering*, March 1995, 194-203.

[Lam98a] A. van Lamsweerde and E. Letier, “Integrating Obstacles in Goal-Driven Requirements Engineering”, *Proc. ICSE-98: 20th Intrnational Conference on Software Enginering*, Kyoto, April 1998.

[Lam98b] A. van Lamsweerde, R. Darimont and E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering", *IEEE Trans. on Software. Engineering*, Special Issue on Inconsistency Management in Software Development, November 1998.

[Lam98c] A. van Lamsweerde and L. Willemet, "Inferring Declarative Requirements Specifications from Operational Scenarios", *IEEE Trans. on Sofware. Engineering*, Special Issue on Scenario Management, December 1998, 1089-1114.

[Lam00a] A. van Lamsweerde and E. Letier, “Handling Obstacles in Goal-Oriented Requirements Engineering”, *IEEE Transactions on Software Engineering*, Special Issue on Exception Handling, 2000.

[Lam00b] A. van Lamsweerde, “Formal Specification: a Roadmap”. In *The Future of Software Engineering*, A. Finkelstein (ed.), ACM Press, 2000.

[Lam00c] A. van Lamsweerde, “Requirements Engineering in the Year 00: A Research Perspective”. Invited Keynote Paper, *Proc. ICSE’2000: 22nd International Conference on Software Engineering*, ACM Press, 2000, pp. 5-19.

[Lee91] J. Lee, "Extending the Potts and Bruns Model for Recording Design Rationale", *Proc. ICSE-13 - 13th Intl. Conf. on Software Engineering*, IEEE-ACM, 1991, 114-125.

[Lei97] J.C. Leite, G. Rossi, F. Balaguer, V. Maiorana, G. Kaplan, G. Hadad and A. Oliveiros, “Enhancing a Requirements Baseline with Scenarios”, *Requirements Engineering Journal* Vol. 2 No. 4, 1997, 184-198.

[Let01] E. Letier, *Reasoning about Agents in Goal-Oriented Requirements Engineering*. Ph. D. Thesis, University of Louvain, May 2001.

[Lev95] N. Leveson, *Safeware - System Safety and Computers*. Addison-Wesley, 1995.

[Man92] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992.

[Man96] Z. Manna and the STep Group, “STeP: Deductive-Algorithmic Verification of Reactive and Real-Time Systems”, *Proc. CAV’96 - 8th Intl. Conf. on Computer-Aided Verification*, LNCS 1102, Springer-Verlag, July 1996, 415-418.

[Mas97] P. Massonet and A. van Lamsweerde, “Analogical Reuse of Requirements Frameworks”, Proc. RE-97 - *3rd Int. Symp. on Requirements Engineering*, Annapolis, 1997, 26-37.

[Mos85] J. Mostow, "Towards Better Models of the Design Process", AI Magazine, Vol. 6, 1985, pp. 44-57.

[Mun81] E. Munford, "Participative Systems Design: Structure and Method", *Systems, Objectives, Solutions*, Vol. 1, North-Holland, 1981, 5-19.

[Myl92] Mylopoulos, J., Chung, L., Nixon, B., “Representing and Using Nonfunctional Requirements: A Process-Oriented Approach”, *IEEE Trans. on Sofware. Engineering*, Vol. 18 No. 6, June 1992, pp. 483-497.

[Myl99] J. Mylopoulos, L. Chung and E. Yu, "From Object-Oriented to Goal-Oriented Requirements Analysis", Communications of the ACM, Vol. 42 No. 1, January 1999, 31-37.

[Nil71] N.J. Nilsson, *Problem Solving Methods in Artificial Intelligence*. McGraw Hill, 1971.

[Nix93] B. A. Nixon, "Dealing with Performance Requirements During the Development of Information Systems", Proc. RE’93 - 1st Intl. IEEE Symp. on Requirements Engineering, Jan. 1993, 42-49.

[Nus94] B. Nuseibeh, J. Kramer and A. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specifications", *IEEE Transactions on Software Engineering*, Vol. 20 No. 10, October 1994, 760-773.

[Par95] D.L. Parnas and J. Madey, “Functional Documents for Computer Systems”, *Science of Computer Programming*, Vol. 25, 1995, 41-61.

[Pot94] C. Potts, K. Takahashi and A.I. Anton, "Inquiry-Based Requirements Analysis", *IEEE Software*, March 1994, 21-32.

[Pot95] C. Potts, “Using Schematic Scenarios to Understand User Needs”, *Proc. DIS’95 - ACM Symposium on Designing interactive Systems: Processes, Practices and Techniques*, University of Michigan, August 1995.

[Rob89] Robinson, W.N., “Integrating Multiple Specifications Using Domain Goals”, *Proc. IWSSD-5 - 5th Intl. Workshop on Software Specification and Design*, IEEE, 1989, 219-225.

[Rol98] C. Rolland, C. Souveyet and C. Ben Achour, “Guiding Goal Modeling Using Scenarios”, *IEEE Trans. on Sofware. Engineering*, Special Issue on Scenario Management, December 1998, 1055-1071.

[Ros77] D.T. Ross and K.E. Schoman, "Structured Analysis for Requirements Definition", *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, 1977, 6-15.

[Rub92] K.S. Rubin and A. Goldberg, "Object Behavior Analysis", Communications of the ACM Vol. 35 No. 9, September 1992, 48-62.

[Som97] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*. Wiley, 1997.

[Sut93] A. Sutcliffe and N. Maiden, “Bridging the Requirements Gap: Policies, Goals and Domains”, *Proc. IWSSD-7 - 7th Intl. Workshop on Software Specification and Design*, IEEE, 1993.

[Sut98] A. Sutcliffe, “Scenario-Based Requirements Analysis”, *Requirements Engineering Journal* Vol. 3 No. 1, 1998, 48-65.

[Swa82] W. Swartout and R. Balzer, "On the Inevitable Intertwining of Specification and Implementation", Communications of the ACM, Vol. 25 No. 7, July 1982, 438-440.

[Yue87] K. Yue, “What Does It Mean to Say that a Specification is Complete?”, *Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design*, Monterey, 1987.

[Yu93] E.S.K. Yu, "Modelling Organizations for Information Systems Requirements Engineering", *Proc. RE'93 - 1st Intl Symp. on Requirements Engineering*, IEEE, 1993, 34-41.

[Yu97] E. Yu, “Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering”, *Proc. RE-97 - 3rd Int. Symp. on Requirements Engineering*, Annapolis, 1997, 226-235.

[Zav97a] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering", *ACM Transactions on Software Engineering and Methodology*, 1997, 1-30.

[Zav97b] P. Zave, “Classification of Research Efforts in Requirements Engineering”, *ACM Computing Surveys*, Vol. 29 No. 4, 1997, 315-321.

ANNEX 1: GOAL REFINEMENT TREE AND RESPONSIBILITY ASSIGNMENT IN THE BART SYSTEM