Requirements Engineering in the Year 00: A Research Perspective

Axel van Lamsweerde

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

ABSTRACT

Requirements engineering (RE) is concerned with the identification of the goals to be achieved by the envisioned system, the operationalization of such goals into services and constraints, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices, and software. The processes involved in RE include domain analysis, elicitation, specification, assessment, negotiation, documentation, and evolution. Getting high-quality requirements is difficult and critical. Recent surveys have confirmed the growing recognition of RE as an area of utmost importance in software engineering research and practice.

The paper presents a brief history of the main concepts and techniques developed to date to support the RE task, with a special focus on modeling as a common denominator to all RE processes. The initial description of a complex safety-critical system is used to illustrate a number of current research trends in RE-specific areas such as goal-oriented requirements elaboration, conflict management, and the handling of abnormal agent behaviors. Opportunities for goal-based architecture derivation are also discussed together with research directions to let the field move towards more disciplined habits.

1. INTRODUCTION

Software requirements have been repeatedly recognized during the past 25 years to be a real problem. In their early empirical study, Bell and Thayer observed that inadequate, inconsistent, incomplete, or ambiguous requirements are numerous and have a critical impact on the quality of the resulting software [Bel76]. Noting this for different kinds of projects, they concluded that *"the requirements for a system do not arise naturally; instead, they need to be engineered and have continuing review and revision"*. Boehm estimated that the late correction of requirements errors could cost up to 200 times as much as correction during such requirements engineering [Boe81]. In his classic paper on the essence and accidents of software engineering, Brooks stated that *"the hardest single part of building a sofware system is deciding precisely what to build... Therefore, the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements"* [Bro87]. In her study of software errors in NASA's Voyager and Galileo programs, Lutz reported that the primary cause of safety-related faults was errors in functional and interface requirements [Lut93].

Recent studies have confirmed the requirements problem on a much larger scale. A survey over 8000 projects undertaken by 350 US companies revealed that one third of the projects were never completed and one half succeeded only partially, that is, with partial functionalities, major cost overruns, and significant delays [Sta95]. When asked about the causes of such failure executive managers identifed poor requirements as the major source of problems (about half of the responses) - more specifically, the lack of user involvement (13%), requirements incompleteness (12%), changing requirements (11%), unrealistic expectations (6%), and unclear objectives (5%). On the European side, a recent survey over 3800 organizations in 17 countries similarly concluded that most of the perceived software problems are in the area of requirements specification (>50%) and requirements management (50%) [ESI96].

Improving the quality of requirements is thus crucial. But it is a difficult objective to achieve. To understand the reason one should first define what requirements engineering is really about.

The oldest definition already had the main ingredients. In their seminal paper, Ross and Schoman stated that *"requirements definition is a careful assessment of the needs 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 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"* [Ros77b]. In other words, requirements engineering must address the contextual goals why a software is needed, the functionalities the software has to accomplish to achieve those goals, and the constraints restricting how the software accomplishing those functions is to be designed and implemented. Such goals, functions and constraints have to be mapped to precise specifications of software behavior; their evolution over time and across software families has to be coped with as well [Zav97b].

This definition suggests why the process of engineering requirements is so complex.

- The scope is fairly broad as it ranges from a world of human organizations or physical laws to a technical artifact that must be integrated in it; from high-level objectives to operational prescriptions; and from informal to formal. The *target system* is not just a piece of *software,*

Invited paper for ICSE'2000 - To appear in
Proc. 22nd International Conference on Software Engineering,
Limerick, June 2000, ACM Press

but also comprises the *environment* that will surround it; the latter is made of humans, devices, and/or other software. The whole system has to be considered under many facets, e.g., socio-economic, physical, technical, operational, evolutionary, and so forth.

- There are multiple concerns to be addressed beside functional ones - e.g., safety, security, usability, flexibility, performance, robustness, interoperability, cost, maintainability, and so on. These non-functional concerns are often conflicting.
- There are multiple parties involved in the requirements engineering process, each having different background, skills, knowledge, concerns, perceptions, and expression means - namely, customers, commissioners, users, domain experts, requirements engineers, software developers, or system maintainers. Most often those parties have conflicting viewpoints.
- Requirement specifications may suffer a great variety of deficiencies [Mey85]. Some of them are errors that may have disastrous effects on the subsequent development steps and on the quality of the resulting software product - e.g., inadequacies with respect to the real needs, incompletenesses, contradictions, and ambiguities; some others are flaws that may yield undesired consequences (such as waste of time or generation of new errors) - e.g., noises, forward references, overspecifications, or wishful thinking.
- Requirements engineering covers multiple intertwined activities.
  - *Domain analysis:* the existing system in which the software should be built is studied. The relevant stakeholders are identified and interviewed. Problems and deficiencies in the existing system are identified; opportunities are investigated; general objectives on the target system are identified therefrom.
  - *Elicitation:* alternative models for the target system are explored to meet such objectives; requirements and assumptions on components of such models are identified, possibly with the help of hypothetical interaction scenarios. Alternative models generally define different boundaries between the software-to-be and its environment.
  - *Negotiation and agreement:* the alternative requirements/assumptions are evaluated; risks are analyzed; "best" tradeoffs that receive agreement from all parties are selected.
  - *Specification:* the requirements and assumptions are formulated in a precise way.
  - *Specification analysis:* the specifications are checked for deficiencies (such as inadequacy, incompleteness or inconsistency) and for feasibility (in terms of resources required, development costs, and so forth).
  - *Documentation:* the various decisions made during the process are documented together with their underlying rationale and assumptions.
  - *Evolution:* the requirements are modified to accommodate corrections, environmental changes, or new objectives.

Given such complexity of the requirements engineering process, rigorous techniques are needed to provide effective support. The objective of this paper is to provide: a brief history of 25 years of research efforts along that way; a concrete illustration of what kind of techniques are available today; and directions to be explored for requirements engineering to become a mature discipline.

The presentation will inevitably be biased by my own work and background. Although the area is inherently interdisciplinary, I will deliberately assume a computing science viewpoint here and leave the socological and psychological dimensions aside (even though they are important). In particular, I will not cover techniques for ethnographic observation of work environments, interviewing, negotiation, and so forth. The interested reader may refer to [Gog93, Gog94] for a good account of those dimensions. A comprehensive, up-to-date survey on the intersecting area of information modeling can be found in [Myl98].

2. THE FIRST 25 YEARS: A FEW RESEARCH MILESTONES

Requirements engineering addresses a wide diversity of domains (e.g., banking, transportation, manufacturing), tasks (e.g., administrative support, decision support, process control) and environments (e.g., human organizations, physical phenomena). A specific domain/task/environment may require some specific focus and dedicated techniques. This is in particular the case for reactive systems as we will see after reviewing the main stream of research..

Modeling appears to be a core process in requirements engineering. The existing system has to be modelled in some way or another; the alternative hypothetical systems have to be modelled as well. Such models serve as a basic common interface to the various activities above. On the one hand, they result from domain analysis, elicitation, specification analysis, and negotiation. On the other hand, they guide further domain analysis, elicitation, specification analysis, and negotiation. Models also provide the basis for documentation and evolution. It is therefore not surprising that most of the research to date has been devoted to techniques for modeling and specification.

The basic questions that have been addressed over the years are:

- what aspects to model in the *why-what-how* range,
- how to model such aspects,
- how to define the model precisely,
- how to reason about the model.

The answer to the first question determines the *ontology* of conceptual units in terms of which models will be built - e.g., data, operations, events, goals, agents, and so forth. The answer to the second question determines the structuring relationships in terms of which such units will be composed and linked together - e.g., input/output, trigger, generalization, refinement, responsibility assignment, and so forth. The answer to the third question determines the informal, semiformal, or formal specification technique used to define the required properties of model components precisely. The answer to the fourth question determines the kind of reasoning technique available for the purpose of elicitation, specification, and analysis.

The early days

The seminal paper by Ross and Schoman opened the field [Ros97b]. Not only did this paper comprehensively explain the scope of requirements engineering; it also suggested goals, viewpoints, data, operations, agents, and resources as potential elements of an ontology for RE. The companion paper introduced SADT as a specific modeling technique [Ros97a]. This technique was a precursor in many respects. It supported multiple models linked through consistency rules - a model for data, in which data are defined by producing/consuming operations; a model for operations, in which operations are defined by input/output data; and a data/operation duality principle. The technique was ontologically richer than many techniques developed afterwards. In addition to data and operations, it supported some rudimentary representation of events, triggering operations, and agents responsible for them. The technique also supported the stepwise refinement of global models into more detailed ones - an essential feature for complex models. SADT was a semi-formal technique in that it could only support the formalization of the declaration part of the system under consideration - that is, what data and operations are to be found and how they relate to each other; the requirements on the data/operations themselves had to be asserted in natural language. The semi-formal language, however, was graphical - an essential feature for model communicability.

Shortly after, Bubenko introduced a modeling technique for capturing entities and events. Formal assertions could be written to express requirements about them, in particular, temporal constraints [Bub80]. At that time it was already recognized that such entities and events had to take part in the real world surrounding the software-to-be [Jac78].

Other semi-formal techniques were developed in the late seventies, notably, entity-relationship diagrams for the modeling of data [Che76], structured analysis for the stepwise modeling of operations [DeM78], and state transition diagrams for the modeling of user interaction [Was79]. The popularity of those techniques came from their simplicity and dedication to one specific concern; the price to pay was their fairly limited scope and expressiveness, due to poor underlying ontologies and limited structuring facilities. Moreover they were rather vaguely defined. People at that time started advocating the benefits of precise and formal specifications, notably, for checking specification adequacy through prototyping [Bal82].

RML brought the SADT line of research significantly further by introducing rich structuring mechanisms such as generalization, aggregation and classification [Gre82]. In that sense it was a precursor to object-oriented analysis techniques. Those structuring mechanisms were applicable to three kinds of conceptual units: entities, operations, and constraints. The latter were expressed in a formal assertion language providing, in particular, built-in constructs for temporal referencing. That was the time where progress in database modeling [Smi77], knowledge representation [Bro84, Bra85], and formal state-based specification [Abr80] started penetrating our field. RML was also probably the first requirements modeling language to have a formal semantics, defined in terms of mappings to first-order predicate logic [Gre86].

Introducing agents

A next step was made by realizing that the software-to-be and its environment are both made of active components. Such components may restrict their behavior to ensure the constraints they are assigned to. Feather's seminal paper introduced a simple formal framework for modeling agents and their interfaces, and for reasoning about individual choice of behavior and responsibility for constraints [Fea87]. 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 a main outcome of the RE process. Once such responsibilities are assigned the agents have contractual obligations they need to fulfill [Fin87, Jon93, Ken93]. Agents on both sides of the software-environment boundary interact through interfaces that may be visualized through context diagrams [War85].

Goal-based reasoning

The research efforts so far were in the *what-how* range of requirements engineering. The requirements on data and operations were just there; one could not capture *why* they were there and whether they were sufficient for achieving the higher-level objectives that arise naturally in any requirements engineering process [Hic74, Mun81, Ber91, Rub92]. Yue was probably the first to argue that the integration of explicit goal representations in requirements models provides a criterion for requirements completeness - the requirements are complete if they are sufficient to establish the goal they are refining [Yue87]. Broadly speaking, a *goal* corresponds to an objective the system should achieve through cooperation of agents in the software-to-be and in the environment.

Two complementary frameworks arose for integrating goals and goal refinements in requirements models: a formal framework and a qualitative one. In the *formal* framework [Dar91], goal refinements are captured through AND/OR graph structures borrowed from problem reduction techniques in artificial intelligence [Nil71]. AND-refinement links relate a goal to a set of subgoals (called refinement); this means that satisfying all subgoals in the refinement is a sufficient condition for satisfying the goal. OR-refinement links relate a goal to an alternative set of refinements; this means that satisfying one of the refinements is a sufficient condition for satisfying the goal. In this framework, a conflict link between goals is introduced when the satisfaction of one of them may preclude the satisfaction of the others. Operationalization links are also introduced to relate goals to requirements on operations and objects. In the *qualitative* framework [Myl92], weaker versions of such link types are introduced to relate "soft" goals [Myl92]. The idea is that such goals can rarely be said to be satisfied in a clear-cut sense. Instead of goal satisfaction, goal satisficing is introduced to express that lower-level goals or requirements are expected to achieve the goal within acceptable limits, rather than absolutely. A subgoal is then said to contribute partially to the goal, regardless of other subgoals; it may contribute positively or negatively. If a goal is AND-decomposed into subgoals and all subgoals are satisficed, then the goal is satisficeable; but if a subgoal is denied then the goal is deniable. If a goal contributes negatively to another goal and the former is satisficed, then the latter is deniable.

The *formal* framework gave rise to the KAOS methodology for eliciting, specifying, and analyzing goals, requirements, scenarios, and responsibility assignments [Dar93]. An optional formal assertion layer was introduced to support various forms of formal reasoning. Goals and requirements on objects are formalized in a real-time temporal logic [Man92, Koy92]; one can thereby prove that a goal refinement is correct and complete, or complete such a refinement [Dar96]. One can also formally detect conflicts among goals [Lam98b] or generate high-level exceptions that may prevent their achievement [Lam98a]. Requirements on operations are formalized by pre-, post-, and trigger conditions; one can thereby establish that an operational requirement “implements” higher-level goals [Dar93], or infer such goals from scenarios [Lam98c].

The *qualitative* framework gave rise to the NFR methodology for capturing and evaluating alternative goal decompositions. One may see it as a cheap alternative to the formal framework, for limited forms of goal-based reasoning, and as a complementary framework for high-level goals that cannot be formalized. The labelling procedure in [Myl92] is a typical example of qualitative reasoning on goals specified by names, parameters, and degrees of satisficing/denial by child goals. This procedure determines the degree to which a goal is satisficed/denied by lower-level requirements, by propagating such information along positive/negative support links in the goal graph.

The strength of those goal-based frameworks is that they do not only cover functional goals but also *non-functional* ones; the latter give rise to a wide range of non-functional requirements. For example, [Nix93] showed how the NFR framework could be used to qualitatively reason about performance requirements during the RE and design phases. Informal analysis techniques based on similar refinement trees were also proposed for specific types of non-functional requirements, such as fault trees [Lev95] and threat trees [Amo94] for exploring safety and security requirements, respectively.

Goal and agent models can be integrated through specific links. In KAOS, agents may be assigned to goals through AND/OR responsibility links; this allows alternative boundaries to be investigated between the software-to-be and its environment. A responsibility link between an agent and a goal means that the agent can commit to perform its operations under restricted pre-, post-, and trigger conditions that ensure the goal [Dar93]. Agent dependency links were defined in [YuM94, Yu97] to model situations where an agent depends on another for a goal to be achieved, a task to be accomplished, or a resource to become available. For each kind of dependency an operator is defined; operators can be combined to define plans that agents may use to achieve goals. The purpose of this modeling is to support the verification of properties such as the viability of an agent's plan or the fulfilment of a commitment between agents.

Viewpoints, facets, and conflicts

Beside the formal and qualitative reasoning techniques above, other work on conflict management has emphasized the need for handling conflicts at the goal level. A procedure was suggested in [Rob89] for identifying conflicts at the requirements level and characterizing them as differences at goal level; such differences are resolved (e.g., through negotiation) and then down propagated to the requirements level. In [Boe95], an iterative process model was proposed in which (a) all stakeholders involved are identified together with their goals (called win conditions); (b) conflicts between these goals are captured together with their associated risks and uncertainties; and (c) goals are reconciled through negotiation to reach a mutually agreed set of goals, constraints, and alternatives for the next iteration.

Conflicts among requirements often arise from multiple stakeholders *viewpoints* [Eas94]. For sake of adequacy and completeness during requirements elicitation it is essential that the viewpoints of all parties involved be captured and eventually integrated in a consistent way. Two kinds of approaches have emerged. They both provide constructs for modeling and specifying requirements from different viewpoints in different notations. In the centralized approach, the viewpoints are translated into some logic-based “assembly” language for global analysis; viewpoint integration then amounts to some form of conjunction [Nis89, Zav93]. In the distributed approach, viewpoints have specific consistency rules associated with them; consistency checking is made by evaluating the corresponding rules on pairs of viewpoints [Nus94]. Conflicts need not necessarily be resolved as they arise; different viewpoints may yield further relevant information during elicitation even though they are conflicting in some respect. Preliminary attempts have been made to define a paraconsistent logical framework allowing useful deductions to be made in spite of inconsistency [Hun98].

Multiparadigm specification is especially appealling for requirements specification. In view of the broad scope of the RE process and the multiplicity of system facets, no single language will ever serve all purposes. Multiparadigm frameworks have been proposed to combine multiple languages in a semantically meaningful way so that different facets can be captured by languages that fit them best. OMT’s combination of entity-relationship, dataflow, and state transition diagrams was among the first attempts to achieve this at a semiformal level [Rum91]. The popularity of this modeling technique and other similar ones led to the UML standardization effort [Rum99]. The viewpoint construct in [Nus94] provides a generic mechanism for achieving such combinations. Attempts to integrate semi-formal and formal languages include [Zav96], which combines state-based specifications [Pot96] and finite state machine specifications; and [Dar93], which combines semantic nets [Qui68] for navigating through multiple models at surface level, temporal logic for the specification of the goal and object models [Man92, Koy92], and state-based specification [Pot96] for the operation model.

Scenario-based elicitation and validation

Even though goal-based reasoning is highly appropriate for requirements engineering, goals are sometimes hard to elicit. Stakeholders may have difficulties expressing them in abstracto. Operational scenarios of using the hypothetical system are sometimes easier to get in the first place than some goals that can be made explicit only after deeper understanding of the system has been gained. This fact has been recognized in cognitive studies on human problem solving [Ben93]. Typically, a *scenario* is a temporal sequence of interaction events between the software-to-be and its environment in the restricted context of achieving some implicit purpose(s). A recent study on a broader scale has confirmed scenarios as important artefacts used for a variety of purposes, in particular in cases when abstract modeling fails [Wei98]. Much research effort has therefore been recently put in this direction [Jar98]. Scenario-based techniques have been proposed for elicitation and for validation - e.g., to elicit requirements in hypothetical situations [Pot94]; to help identify exceptional cases [Pot95]; to populate more abstract conceptual models [Rum91, Rub92]; to validate requirements in conjunction with prototyping [Sut97], animation [Dub93], or plan generation tools [Fic92]; to generate acceptance test cases [Hsi94].

The work on deficiency-driven requirements elaboration is especially worth pointing out. A system there 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 technique is twofold: (a) detect inconsistencies between scenarios and goals; (b) apply operators that modify the specification to remove the inconsistencies. Step (a) is carried out by a planner that searches for scenarios leading to some goal violation. (Model checkers might probably do the same job in a more efficient way [McM93, Hol97, Cla99].) 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, and so forth. The repeated application of deficiency detection and debugging operators allows the analyst to explore the space of alternative models and hopefully converge towards a satisfactory system specification.

The problem with scenarios is that they are inherently partial; they raise a coverage problem similar to test cases, making it impossible to verify the absence of errors. Instance-level trace descriptions also raise the combinatorial explosion problem inherent to the enumeration of combinations of individual behaviors. Scenarios are generally procedural, thus introducing risks of overspecification. The description of interaction sequences between the software and its environment may force premature choices on the precise boundary between them. Last but not least, scenarios leave required properties about the intended system implicit, in the same way as safety/liveness properties are implicit in a program trace. Work has therefore begun on inferring goal/requirement specifications from scenarios in order to support more abstract, goal-level reasoning [Lam98c].

Back to groundwork

In parallel with all the work outlined above, there has been some more fundamental work on clarifying the real nature of requirements [Jac95, Par95, Zav97]. This was motivated by a certain level of confusion and amalgam in the literature on requirements and software specifications. At about the same time, Jackson and Parnas independently made a first important distinction between *domain properties* (called indicative in [Jac95] and NAT in [Par95]) and requirements (called optative in [Jac95] and REQ in [Par95]). Such distinction is essential as physical laws, organizational policies, regulations, or definitions of objects or operations in the environment are by no means requirements. Surprisingly, the vast majority of specification languages existing to date do not support that distinction. A second important distinction made by Jackson and Parnas was between (system) requirements and (software) specifications. *Requirements* are formulated in terms of objects in the real world, in a vocabulary accessible to stakeholders [Jac95]; they capture required relations between objects in the environment that are monitored and controlled by the software, respectively [Par95]. *Software specifications* are formulated in terms of objects manipulated by the software, in a vocabulary accessible to programmers; they capture required relations between input and output software objects. *Accuracy goals* are non-functional goals requiring that the state of input/output software objects accurately reflect the state of the corresponding monitored/controlled objects they represent [Myl92, Dar93]. Such goals often are to be achieved partly by agents in the environments and partly by agents in the software. They are often overlooked in the RE process; their violation may lead to major failures [LAS93, Lam2Ka]. A further distinction has to be made between requirements and assumptions. Although they are both optative, *requirements* are to be enforced by the software whereas *assumptions* can be enforced by agents in the environment only [Lam98b]. If *R* denotes the set of requirements, *As* the set of assumptions, *S* the set of software specifications, *Ac* the set of accuracy goals, and *G* the set of goals, the following satisfaction relations must hold:

$$\mathrm{S, Ac, D} \models \mathrm{R} \quad \text{with} \quad \mathrm{S, Ac, D} \not\models \textbf{false}$$

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

The reactive systems line

In parallel with all the efforts discussed above, a dedicated stream of research has been devoted to the specific area of reactive systems for process control. The seminal paper here was based on work by Heninger, Parnas and colleagues while reengineering the flight software for the A-7 aircraft [Hen80]. The paper introduced SCR, a tabular specification technique for specifying a reactive system by a set of parallel finite-state machines. Each of them is defined by different types of mathematical functions represented in tabular format. A mode transition table defines a mode (i.e. a state) as a transition function of a mode and an event; an event table defines an output variable (or auxiliary quantity) as a function of a mode and an event; a condition table defines an output variable (or auxiliary quantity) as a function of a mode and a condition (the latter may refer to input or output variables, modes, or auxiliary quantities). The strength of SCR is its use of terminology and tabular notations familiar to domain experts. Although it is lightweight the notation is sufficiently formal to enable useful consistency and completeness checks, based on the property that tables must represent total functions. Last but not least, the technique is now supported by an impressive toolset offering a wide range of analysis - e.g., dedicated consistency/completeness checking, animation, model checking, and theorem proving [Heit96, Heit98a, Heit98b]. The main weakness of SCR is its lack of structuring mechanisms for structuring variables (e.g., by aggregation or generalization), modes (e.g., by AND/OR decomposition), and tables (e.g., by refinement relationships).

Data structuring was provided by CORE [Fau92], a variant of SCR supporting some form of object orientation. The work around Statecharts [Har87, HAR96] showed how state machine specifications could be recursively AND/OR decomposed into finer ones so as to support a stepwise specification refinement process. The specification language is fully graphical and sufficiently formal to enable powerful animation tools [Har90]. But formality (and therefore analysis) is more limited than SCR. The work on RSML has taken one step further by extending Statecharts with interface descriptions and direct communication among parallel state machines; state transitions are more precisely defined [Lev94]. As a result, the same range of analysis as SCR can be provided with structuring facilities in addition [Heim96, Cha98, Tho99]. The RSML language is still graphical and integrates tabular formats as well. Like SCR, the technique has been validated by experience in complex projects - notably, the documentation of the specifications of TCAS II, a Traffic Collision Avoidance System required on all commercial aircrafts flying in US airspace [Lev94].

Requirements reuse

Requirements refer to specific domains and to specific tasks. Requirements within similar domains and/or for similar tasks are more likely to be similar than the software components implementing them. Surprisingly enough, techniques for retrieving, adapting, and consolidating reusable requirements have received relatively little attention in comparison with all the work on software reuse. The area was initiated by [Reu91] in which a technique based on inheritance was proposed to reuse fragments of domain descriptions (e.g. in the library domain) and of task specifications (e.g., history tracking). Analogical and case-based reasoning techniques have been borrowed from artificial intelligence to support structural matching [Mai93] and semantic matching [Mas97] in the requirements retrieval process. On the task reuse side, the work on problem frames reprsents a preliminary attempt to classify and characterize task patterns [Jac95].

The work in this area has not made sufficient progress to date to determine whether such approaches may be practical and may scale up.

Requirements documentation

The specifications of the domain and requirements models are essential components to document requirements for communication, inspection, negotiation, and evolution. Ideally they should only be part of it. Some work has been done on capturing the process and rationale leading to such models [Sou93, Nus94] and the actors responsible for decisions so that traceability links can be established [Got95].

3. FROM OBJECT ORIENTATION TO GOAL ORIENTATION

Today's object-oriented analysis techniques have a strong impact on the state of practice in requirements engineering. As introduced before, they combine multiple semi-formal modeling techniques to capture different facets of the system (such as the data, behavioral, and interaction facets); they provide structuring mechanisms (such as generalization and aggregation); they offer a wide spectrum of notations that can be used from requirements modeling to design (at some risk of confusion between those phases); they now tend towards a standard set of notations [Rum99], with built-in extension mechanisms, which hopefully will in the end have a precise semantics. However, the concepts and structuring mechanisms supported essentially emerged by abstraction from the programming field [Myl99] - the same way as structured analysis came out by abstraction from structured programming techniques. In particular, the *why* concerns in the early stages of requirements engineering practice [Hic74, Ros77b, Mun81, Ber91] are not addressed.

The aim of this section is to illustrate the benefits of looking the other way round for the purpose of requirements elicitation, specification, and analysis - that is, to start thinking about objectives as they arise in preliminary material provided, use goal refinement/abstraction as higher-level mechanism for model/specification structuring, and thereby incrementally derive multiple models:

- the goal model, leading to operational requirements;
- the object model;
- the agent responsibility model, leading to alternative system boundaries to be explored;
- the operation model.

To suggest that goal-based reasoning is not only useful in the context of enterprise modeling, we take a recent benchmark proposed to the formal specification community: the BART system [BAR99]. This case study is appealling for a number of reasons: it is a real system; it is a complex, real-time, safety-critical system; the initial document was provided by an independent source involved in the development. The model elaboration will inevitably be sketchy due to lack of space. We select a few snaphots from the KAOS elaboration that mix informal, semi-formal, and formal specifications. More details can be found in [Let2K].

The initial document [BAR99] 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.

Goal identification from the initial document

Figure 1 gives a portion of the goal graph identified after a first reading of the initial document. The goals were obtained by searching for intentional keywords such as “purpose”, “objective”, “concern”, “intent”, “in order to”, and so forth. In this graphical specification, clouds denote soft goals (used in general to select among alternatives), parallelograms denote formalizable goals, arrows denote goal-subgoal links, a double line linking arrows denotes an OR-refinement into alternative subgoals, and a crossed link denotes a conflict. 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 “→“ connective denotes logical implication; $\Box\ (P \rightarrow Q)$ is denoted by $P \Rightarrow Q$ for short.

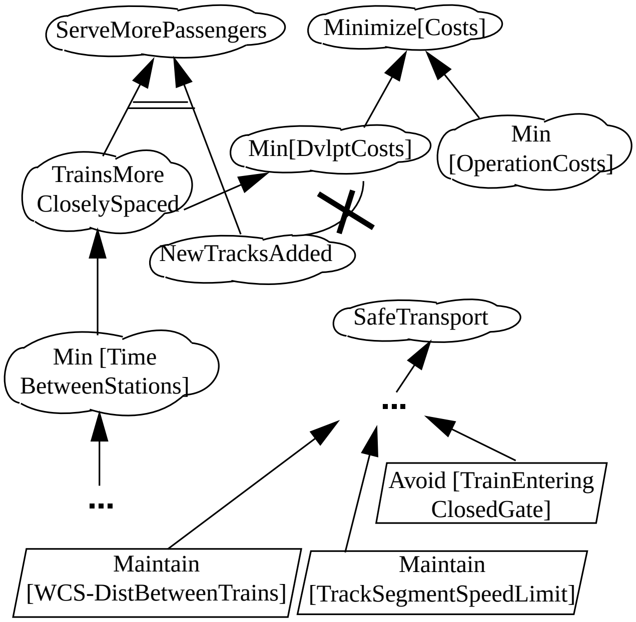


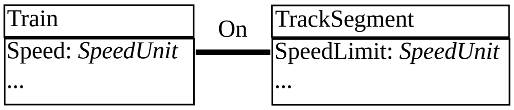
Figure 1 - Preliminary goal graph for the BART system

Formalizing goals and identifying objects

As safety goals are critical one may start thinking about them first. The goal Maintain[TrackSegmentSpeedLimit] at the bottom of Figure 1 may be defined more precisely:

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

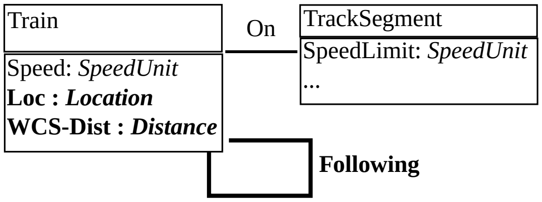
The predicate, objects, and attributes appearing in this goal formalization give rise to the following portion of the object model:



The other goal at the bottom of Figure 1 is defined precisely as well:

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 :
$Following(tr1, tr2) \Rightarrow tr1.Loc - tr2.Loc > tr1.WCS\text{-}Dist$

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. The initial portion of the object model is now enriched from that second goal definition:



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.

Eliciting new goals through WHY questions

It is often worth eliciting more abstract goals than those easily identifiable from the initial document (or from interviews). The reason is that one may thereby find out other important subgoals of the more abstract goal that were overlooked in the first place.

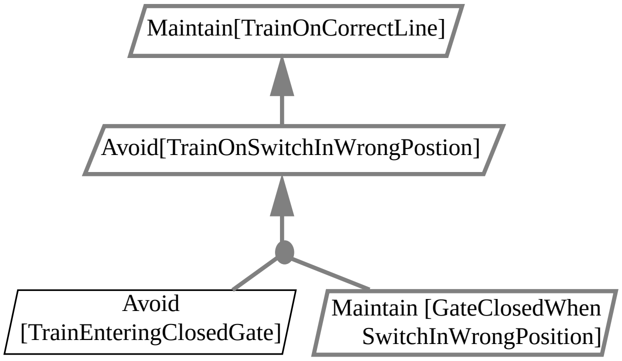

Maintain[TrainOnCorrectLine]
Avoid[TrainOnSwitchInWrongPostion]
Avoid [TrainEnteringClosedGate]
Maintain [GateClosedWhen SwitchInWrongPosition]

Figure 2 - Enriching the goal graph by WHY elicitation

More abstract goals are identified by asking WHY questions. 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, Let2K]. The dot joining the two lower refinement links together in Figure 2 means that the refinement is (provably) complete.

The quest of more abstract goals should of course remain within the system's subject matter [Zav97a].

Eliciting new goals through HOW questions

Goals have 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 in the former case become requirements; they are 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.

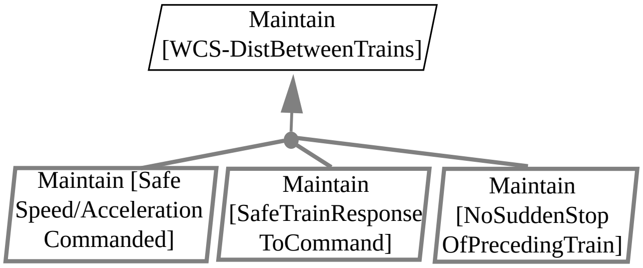

Maintain [WCS-DistBetweenTrains]
Maintain [Safe Speed/Acceleration Commanded]
Maintain [SafeTrainResponse ToCommand]
Maintain [NoSuddenStop OfPrecedingTrain]

Figure 3 - Goal refinement

The formalization of the three subgoals in Figure 3 may be used to prove that together they entail the father goal Maintain[WCS-DistBetweenTrains] formalized before [Let2K]. These subgoals have to 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[SafeTrainResponseToCommand] 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 OnBoardTrainController agent. This alternative would give rise to a fully distributed system.

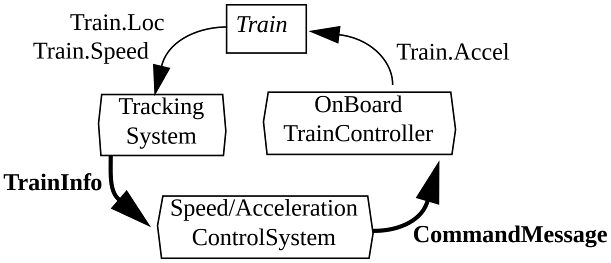
To help making choices among alternatives, qualitative reasoning techniques might be applied to the softgoals identified in Figure 1 [Myl99].

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 $\forall$ cm: *CommandMessage*, ti1, ti2: *TrainInfo*
cm.Sent $\wedge$ cm.TrainID = ti1.TrainID $\wedge$ FollowingInfo (ti1, ti2)
$\Rightarrow$ cm.Accel $\leq$ F (ti1, ti2) $\wedge$ cm.Speed $>$ G (ti1)

To fulfil its responsibility for this goal the Speed/AccelerationControlSystem agent must be able to evaluate the goal antecedent and establish the goal consequent. The agent's monitored object is therefore TrainInfo whereas its controlled variables are CommandMessage.Accel and CommandMessage.Speed. The following agent interfaces are derived by this kind of reasoning:



Identifying operations

Goals refer to specific state transitions; for each of them an operation causing it is identified and preliminarily defined by domain pre- and postconditions that 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 $\neg$ cm.Sent
DomPost cm.Sent $\wedge$ 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 purpose of the operationalization 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. Preliminary derivation rules for an operationalization calculus were introduced in [Dar93]. 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) $\wedge$ Following (ti1, ti2)
$\rightarrow$ cm.Acc $\leq$ F (ti1, ti2) $\wedge$ cm.Speed $>$ G (ti1)
ReqTrig for CmdMsgSentInTime:
$\blacksquare_{\leq 0.5 \text{ sec}}$ $\neg \exists$ cm2: CommandMessage:
cm2.Sent $\wedge$ 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. To sum up, goal-oriented requirements engineering has many advantages:

- object models and requirements are derived systematically from goals,
- goals provide the rationale for requirements,
- 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.

4. LIVING WITH CONFLICTS

As discussed earlier in the paper, goals also provide a firm basis for conflict analysis. Requirements engineers live in a world where conflicts are the rule, not the exception [Eas94]. Conflicts must be detected and eventually resolved even though they may temporarily be useful for eliciting further information.

The initial BART document suggests an interesting example of conflict [BAR99, p.13]. Figure 4 helps visualizing it.

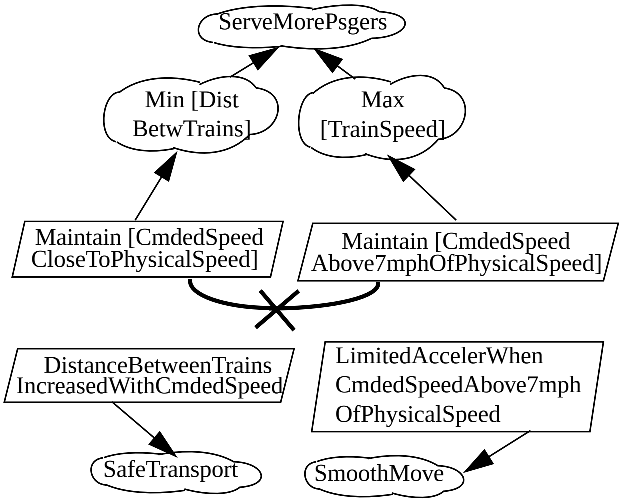

ServeMorePsgers
Min [Dist BetwTrains]
Max [TrainSpeed]
Maintain [CmdedSpeed CloseToPhysicalSpeed]
Maintain [CmdedSpeed Above7mphOfPhysicalSpeed]
DistanceBetweenTrains IncreasedWithCmdedSpeed
LimitedAccelerWhen CmdedSpeedAbove7mph OfPhysicalSpeed
SafeTransport
SmoothMove

Figure 4 - Conflict in speed/acceleration control

Roughly speaking, the commanded speed may not be too high, because otherwise it forces the distance between trains to be too high for safety reason (see the left part of Figure 4); on the other hand, the commanded speed may not be too low, because otherwise it may force uncomfortable acceleration (see the right part of Figure 4). To be more precise, we look at the formalizations produced during goal elaboration:

Goal Maintain [CmdedSpeedCloseToPhysicalSpeed]
FormalDef $\forall$ tr: Train

$$tr.Acc_{CM} \geq 0$$
$$\Rightarrow tr.Speed_{CM} \leq tr.Speed + f\,(\text{dist-to-obstacle})$$

and

Goal Maintain [CmdedSpeedAbove7mphOfPhysicalSpeed]
FormalDef $\forall$ 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

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

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 $\forall$ tr: Train

$$tr.Acc_{CM} \geq 0 \Rightarrow tr.Speed_{CM} > tr.Speed + 7$$
$$\vee\ \mathit{f\,(dist\text{-}to\text{-}obstacle) \leq 7}$$

5. BEING PESSIMISTIC

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

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. To illustrate this, consider some of the goals appearing at the bottom of the refinement tree in Annex 1.

The goal Achieve[CmdMsgSentInTime] may be obstructed by conditions such as:

CommandNotSent,
CommandSentLate,
CommandSentToWrongTrain

The goal Maintain[SafeCmdMsg] may be obstructed by the condition

UnsafeAcceleration,

and so on. We call such obstructing conditions *obstacles* [Pot95]. Obstacles can be produced for each goal by constructing a goal-anchored fault-tree, that is, a refinement tree whose root is the goal negation. Formal and heuristic techniques are available for generating obstacles systematically from goal specifications and domain properties [Lam2Ka].

Alternative resolution strategies may then be applied to the generated obstacles in order to produce new or alternative requirements. For example, the obstacle CommandSentLate 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 [BAR99]; 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, Maintain[NoCollisionWhenOutOfDateTrainInfo]. This new goal has to be refined in turn, e.g., by subgoals requiring full braking when the message origination time tag has expired.

6. FROM REQUIREMENTS TO ARCHITECTURE

Currently there is very little support for building or modifying a software architecture guaranteed to meet a set of functional and non-functional requirements. Proposals for architectural description languages and associated analysis techniques have flourished [Luc95, Mag95, Tay96, Gar97]; constructive techniques have also been proposed for architectural refinement [Mor95]. However, little work has been devoted to date to techniques for systematically deriving architectural descriptions from requirements specifications. This is somewhat paradoxical as the software architecture has long been recognized to have a profound impact on the achievement of non-functional goals such as security, availability, fault tolerance, evolvability, and so forth [Per92, Sha96].

A goal-based approach for architecture derivation might be useful and is feasible. The general principle is to:

- use functional goals assigned to software agents to derive a first abstract dataflow architecture,
- use non-functional goals to refine dataflow connectors.

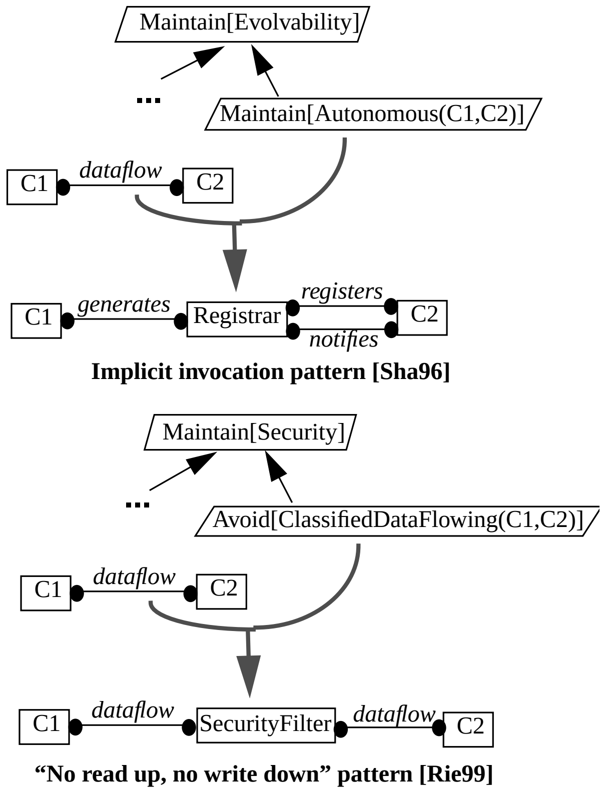
The first step is rather simple; once a software agent is assigned to a functional goal its interfaces in terms of monitored/controlled variables can be determined systematically (see Section 5). The agents become architectural components; the dataflow connectors are then derived from input/output data dependencies. (The granularity of such components is determined by the granularity of goal refinement.)

The second step is the difficult one. There is some hope here that connector refinement patterns could be used to support the process. The idea is to annotate such patterns with non-functional goals they achieve, and to consider applying a pattern when its associated goal matches the goal under consideration. A catalog of patterns would codify the architect's knowledge [Mor95] - much the same way as [Gam95] but at the architecting level and with a proof (or a solid argument), once for all, that the associated goal is established.

Figure 5 sketches a few such patterns to help visualizing the general idea.

Preliminary experience with this approach on small examples suggests that it is worth investigating further. In particular, refinement patterns must be combined with abstraction patterns to be applied to components from the implementation infrastructure imposed.

Explicit links between refined connectors and non-functional goals would also allow architectural views to be extracted through queries (e.g., security view, availability view, etc.).



Implicit invocation pattern [Sha96]

“No read up, no write down” pattern [Rie99]

Figure 5 - Goal-driven connector refinement

7. MORE WORK FOR THE NEXT 25 YEARS!

Efforts should thus be devoted to bridging the gap between RE research and research in software architecture. Even though streamlined derivation processes may be envisaged for software development, things get much more complicated for software evolution. For example, the conflict between requirements volatility and architectural stability is a difficult one to handle.

In some application domains, complex customizable packages are increasingly often chosen by clients as an alternative to software development. Another unexplored transition that should be investigated is the systematic derivation of parameter settings from requirements.

Massive access to the internet will enable more and more end-users to access software applications. Define-it-yourself approaches should therefore be explored to support RE-in-the-small involving end-users as the only stakeholders.

The gap between RE research and formal specification research is another important one to bridge. Roughly speaking, the former offers much richer modeling abstractions while the latter offers much richer analysis - such as model checking, deductive verification, animation, test data generation, formal reuse of components, or refinement from specification to implementation [Lam2Kb]. The technology there is reaching a level of maturity where tool prototypes evolve into professional products and impressive experience in fully formal development of complex systems is emerging [Beh99]. One should therefore look at ways for mapping the conceptually richer world of RE to the formal analysis world. One recent attempt in this general direction is worth pointing out [Dwy99].

Domain and requirements models should ideally capture more knowledge about the multiple aspects, concerns, and activities involved in the requirements engineering process. The problem here is to find best compromises between model expressiveness and precision, for richer analysis, and model simplicity, for better usability. In particular, one should look at effective combinations that integrate semi-formal, formal, and qualitative reasoning about non-functional requirements.

Modeling agents is a particular area of concern. Traditional RE has decomposed the world in two components - the software and its environment. Most often there are multiple software, human and physical components having to cooperate. Limited capabilities, inaccurate beliefs, poor cooperation, and wrong assumptions may be sources of major problems [LAS93, Lev95, But98]. Much work is needed here to support agent-based reasoning during requirements elaboration and, in particular, responsibility assignment.

Models for reasoning about current alternatives and future plausible changes have received relatively little attention to date. Such reasoning should be at the heart of the RE process though. These are exciting fields open for exploration.

Much RE work has been done on new languages and sets of notations. It is time to shift towards building complex artefacts using such languages. *Constructive* techniques are needed to guide requirements engineers in the incremental elaboration and assessment of requirements. In particular, one should clarify when and where to shift from informal through semi-formal to formal; when and how to shift from scenarios to requirements models; when and how to shift from conflicting viewpoints to a consistent documentation; and so forth.

Another area of investigation is requirements reengineering. It frequently happens that existing requirements documents are so poorly written and structured that it is hard to work with them later on during development and maintenance. Abstraction and restructuring techniques would be highly useful in this context.

On the language side itself, one should care more for semantically meaningful integrations of multiple languages to capture the multiple facets of the system; manipulation of multiple formats for the same language (e.g., textual, tabular, graphical); and multibutton analysis where different levels of optional analysis are provided - from cheap, surface-level to more expensive, deep-level [Lam2Kb].

On the tool side, there are many opportunities for RE-specific developments. Let us suggest just a few examples. The final deliverable of the requirements phase is most often a document in natural language that in addition to indicative and optative statements may integrate graphical portions of models, excerpts from interviews, and so forth. A most welcome tool would be one to assist in the generation of such a document to keep the structure of the requirements model (e.g., the goal refinement structure), extract relevant portions of it, and maintain traceability links to subsidiary elicitation material. Earlier in the RE process, one might envisage dynamic tools for exploration of alternatives that like games unfold based on the actions of users and integrate a variety of interactive presentation media - e.g., interview video, originals of documentation and so on [Fea97]. A last example are tools for supporting requirements evolution through runtime monitoring and resolution of deviations between the system's behavior and its original requirements [Fea98].

8. BY WAY OF CONCLUSION

The last 25 years have seen growing interest and efforts towards ameliorating the critical process of engineering higher-quality requirements. We have reviewed a number of important milestones along that way and tried to convince the reader that goal-based reasoning is central to requirements engineering - for requirements elaboration, exploration of alternative software boundaries, conflict management, requirements-level exception handling, and architecture derivation. Goals are also abstractions stakeholders are familiar with. In all the industrial projects our technology transfer institute has been involved in, it turned out that high-level managers and decision makers were much more interested in checking goal models than, e.g., object models.

We also tried to suggest that much remains to be done. The work is worth the effort though. After all, given the expected progress in component reuse and automated programming technologies, will there be anything else left in software engineering, beside software geriatry, than requirements engineering?

Acknowledgment.

Emanuel Letier was instrumental in developing the KAOS specification of the BART system from which the excerpts in this paper are taken. I am also grateful to the people at CEDITI who are using some of the ideas presented here in industrial projects; special thanks are due to Robert Darimont and Emmanuelle Delor for regular feedback from their experience. Numerous discussions with Martin Feather, Steve Fickas, and John Mylopoulos have influenced some of the views presented in this paper.

REFERENCES

[Abr80] J.R. Abrial, “The Specification Language Z: Syntax and Semantics”. Programming Research Group, Oxford Univ., 1980.

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

[Bal82] R.M. Balzer, N.M. Goldman, and D.S. Wile, “Operational Specification as the Basis for Rapid Prototyping”, *ACM SIGSOFT Softw. Eng. Notes* Vol. 7 No. 5, Dec. 1982, 3-16.

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

[Beh99] P. Behm, P. Benoit, A. Faivre and J.M. Meynadier, “Météor: A Successful Application of B in a Large Project”, *Proc. FM-99 - World Conference on Formal Methods in the Development of Computing Systems*, LNCS 1708, Springer-

Verlag, 1999, 369-387.

[Bel76] T.E. Bell and T.A. Thayer, “Software Requirements: Are They Really a Problem?”, *Proc. ICSE-2: 2nd Intrnational Conference on Software Enginering,* San Francisco, 1976, 61-68.

[Ben93] K. M. Benner, M. S. Feather, W. L. Johnson and L. A. Zorman, “Utilizing Scenarios in the Software Development Process”, *Information System Development Process*, Elsevier Science Publisher B.V. (North-Holland), 1993, 117-134.

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

[Boe81] B.W. Boehm, Software Engineering Economics. Prentice-Hall, 1981.

[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.

[Bra85] R.J. Brachman and H.J. Levesque (eds.), Readings in Knowledge Representation, Morgan Kaufmann, 1985.

[Bro84] M. Brodie, J. Mylopoulos, and J. Schmidt (eds.), *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*. Springer-Verlag, 1984.

[Bro87] F.P. Brooks "No Silver Bullet: Essence and Accidents of Software Engineering". *IEEE Computer*, Vol. 20 No. 4, April 1987, pp. 10-19.

[Bub80] J. Bubenko, “Information Modeling in the Context of System Development”, *Proc. IFIP Congress ‘80*, North Holland, 1980, 395-411.

[But98] R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, "A Formal Methods Approach to the Analysis of Mode Confusion", Proceedings DASC'98 - 17th Digital Avionics Systems Conference, Seattle, November 1998. See also http://shemesh.larc.nasa.gov/fm/fm-now-mode-confusion.html.

[Cha98] W. Chan, R. Anderson, P. Beame, S. Burns F. Modugno, D. Notkin, and J. Reese, “Model Checking Large Software Specifications”, *IEEE Transactions on Software Engineering* Vol. 24 No. 7, 1998, 498-520.

[Che76] P. Chen, “The Entity Relationship Model - Towards a Unified View of Data”, ACM Transactions on Database Systems, vol. 1, no. 1, March 1976, 9-36.

[Cla99] E.M. Clarke, O. Grumberg, and D.A. Peled, *Model Checking*. MIT Press, 1999.

[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.

[Dem78] T. DeMarco, Structured Analysis and System Specification, Yourdon Press, 1978.

[Dub93] E. Dubois, Ph. Du Bois and M. Petit, "Object-Oriented Requirements Analysis: An Agent Perspective", Proc. ECOOP’93 - 7th European Conf. on Object-Oriented Programming, Springer-Verlag LNCS 707, 1993, 458-481.

[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 Engineering*, 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.

[ESI96] European Software Institute, "European User Survey Analysis", Report USV_EUR 2.1, ESPITI Project, January 1996.

[Fau92] S. Faulk, J. Brackett, P. Ward and J. Kirby, “The CORE Method for Real-Time Requirements”, *IEEE Software*, September 1992, 22-33.

[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.

[Fea97] M. Feather, S. Fickas, A. Finkelstein, and A. van Lamsweerde, “Requirements and Specification Exemplars”, *Automated Software Engineering* Vol. 4, 1997, 419-438.

[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.

[Gam95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns — Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Gar97] D. Garlan, R. Monroe and D. Wile, "ACME: An Architecture Description Interchange Language", Proceedings CASCON'97, Toronto, Nov. 1997, 169-183.

[Gog93] J. Goguen and C. Linde, “Techniques for Requirements Elicitation”, *Proc. RE’93 - First IEEE Symposium on Requirements Engineering*, San Diego, 1993, 152-164.

[Gog94] J. Goguen and M. Jirotka (eds.), *Requirements Engineering: Social ad Technical Issues*. Academic Press, 1994.

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

[Gre82] S.J. Greenspan, J. Mylopoulos, and A. Borgida, “Capturing More World Knowledge in the Requirements Specification”, *Proc. ICSE-6: 6th Intrnational Conference on Software Enginering,* Tokyo, 1982.

[Gre86] S.J. Greenspan, A. Borgida and J. Mylopoulos, “A Requirements Modeling Language and its Logic”, *Information Systems* Vol. 11 No. 1, 1986, 9-23.

[Har87] D. Harel, “Statecharts: A Visual Formalism for Complex Systems”, *Science of Computer Programming* Vol. 8, 1987, 231-274.

[Har90] D.Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R.

Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, “STATEMATE: A Working Environment for the Development of Complex Reactive Systems”, *IEEE Transactions on Software Engineering*, Vol. 16 No. 4, April 1990, 403-414.

[Har96] D. Harel, “The STATEMATE Semantics of Statecharts”, *ACM Transactions on Software Engineering and Methodology* Vol. 5 No.4, 1996, 293-333.

[Heim96] M.P. Heimdahl and N.G. Leveson, "Completeness and Consistency in Hierarchical State-Based Requirements", *IEEE Transactions on Software Engineering* Vol. 22 No. 6, June 1996, 363-377.

[Heit96] C. Heitmeyer, R. Jeffords and B. Labaw, "Automated Consistency Checking of Requirements Specificatons", *ACM Transactions on Software Engineering and Methodology* Vol. 5 No. 3, July 1996, 231-261.

[Hei98a] C. Heitmeyer, J. Kirkby, B. Labaw, M. Archer and R. Bharadwaj, “Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications”, *IEEE Transactions on Software Engineering* Vol. 24 No. 11, November 1998, 927-948.

[Hei98b] C. Heitmeyer, J. Kirkby, B. Labaw, and R. Bharadwaj, “SCR*: A Toolset for specifying and Analyzing Software Requirements”, *Proc. CAV’98 - 10th Annual Conference on Comuter-Aided Verification*, Vancouver, 1998, 526-531.

[Hen80] K.L. Heninger, “Specifying Software Requirements for Complex Systems: New Techniques and their Application”, *IEEE Transactions on Software Engineering* Vol. 6 No. 1, January 1980, 2-13.

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

[Hol97] G. Holzman, "The Model Checker SPIN", *IEEE Trans. on Software Engineering* Vol. 23 No. 5, May 1997, 279-295.

[Hsi94] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima and C. Chen, “Formal approach to scenario analysis”, *IEEE Software*, Mar 1994, 33-41.

[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.

[Jac78] M.A. Jackson, “Information Systems: Modeling, Sequencing, and Transformation”, *Proc. ICSE-3: 3rd Intrnational Conference on Software Enginering*, Munich, 1978, 72-81.

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

[Jaco93] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard, *Object-Oriented Software Engineering. A Use Case Driven Approach.* Addison-Wesley, ACM Press, 1993.

[Jar98] M.Jarke and R. Kurki-Suonio (eds.), Special Issue on Scenario Management, *IEEE Trans. on Sofware. Engineering*, December 1998.

[Jon93] A.J. Jones and M. Sergot, “On the Characterization of Law and Computer Systems: the Normative System Perspective”, *in* J.Ch. Meyer and R.J. Wieringa (Eds.), *Deontic Logic in Computer Science - Normative System Specification*, Wiley, 1993.

[Ken93] S. Kent, T. Maibaum and W. Quirk, "Formally Specifying Temporal Constraints and Error Recovery", *Proc. RE’93 - 1st Intl. IEEE Symp. on Requirements Engineering*, Jan. 1993, 208-215.

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

[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 Sofware. 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.

[Lam2Ka] A. van Lamsweerde and E. Letier, “Handling Obstacles in Goal-Oriented Requirements Engineering”, *IEEE Transactions on Software Engineering*, Special Issue on Exception Handling, 2000.

[Lam2Kb] A. van Lamsweerde, “Formal Specification: a Roadmap”. In *The Future of Software Engineering*, A. Finkelstein (ed.), ACM Press, 2000.

[LAS93] Report of the Inquiry Into the London Ambulance Service, February 1993. The Communications Directorate, South West Thames Regional Authority, ISBN 0-905133-70-6. *See also* the London Ambulance System home page, http://hsn.lond-amb.sthames.nhs.uk/http.dir/service/organisation/featurs/info.html.

[Let2K] E. Letier and A. van Lamsweerde, “KAOS in Action: the BART System”. IFIP WG2.9 meeting, Flims, http://www.cis.gsu.edu/~wrobinso/ifip2_9/Flims00.

[Lev94] N.G. Leveson, M.P. Heimdahl and H. Hildtreth, “Requirements Specification for Process-Control Systems”, *IEEE Transactions on Software Engineering* Vol. 20 No. 9, September 1994, 684-706.

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

[Luc95] D. Luckham and J. Vera, "An Event-Based Architecture Definition Language", IEEE Transactions on Software Engineering, Vol. 21 No. 9, Sept. 1995, 717-734.

[Lut93] R.R. Lutz, "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems", *Proceedings RE'93 - First International Symposium on Requirements Engineering*, San Diego, IEEE, 1993, 126-133.

[Mag95] J. Magee, N Dulay, S. Eisenbach and J Kramer, "Specifying Distributed Software Architectures", Proceedings ESEC'95 - 5th European Software Engineering Conference, Sitges, LNCS 989, Springer-Verlag, Sept. 1995, 137-153.

[Mai93] N.Maiden and A. Sutcliffe, “Exploiting Reusable Specifications Through Analogy”, *Communications of the ACM* Vol. 35 No. 4, 1993, 55-64.

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

[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.

[McM93] K.L. McMillan, *Symbolic Model Checking: An Approach to the State Explosion Problem*, Kluwer, 1993.

[Mey85] B. Meyer, “On Formalism in Specifications”, *IEEE Software*, Vol. 2 No. 1, January 1985, 6-26.

[Mor95] M. Moriconi, X. Qian, and R. Riemenschneider, "Correct Architecture Refinement", IEEE Transactions on Software Engineering, Vol. 21 No. 4, Apr. 1995, 356-372.

[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.

[Myl98] J. Mylopoulos, "Information Modeling in the Time of the Revolution", Invited Review, *Information Systems* Vol. 23 No. 3/4, 1998, 127-155.

[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.

[Nis89] C. Niskier, T. Maibaum and D. Schwabe, “A Pluralistic Knowledge-Based Approach to Software Specification”, *Proc. ESEC-89 - 2nd European Software Engineering Conference*, LNCS 387, September 1989, 411-423.

[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.

[Per92] D. Perry and A. Wolf, "Foundations for the Study of Software Architecture", ACM Software Engineering Notes, Vol. 17 No. 4, October 1992, 40-52.

[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.

[Pot96] B. Potter, J. Sinclair and D. Till, *An Introduction to Formal Specification and Z.* Second edition, Prentice-Hall, 1996.

[Qui68] R. Quillian, Semantic Memory. In *Semantic Information Processing*, M. Minsky (ed.), MIT Press, 1968, 227-270.

[Reu91] H.B. Reubenstein and R.C. Waters, “The Requirements Apprentice: Automated Assistance for Requirements Acquisition”, *IEEE Transactions on Software Engineering*, Vol. 17 No. 3, March 1991, 226-240.

[Rie99] R.A. Riemenschneider, “Checking the Correctness of Architectural Transformation Steps via Proof-Carrying Architectures”, *Proc. W1CSA1 - First IFIP Conference on Software Architecture*, San Antonio, February 1999.

[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.

[Ros77a] D.T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas", *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, 1977, 16-34.

[Ros77b] 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.

[Rum91] J. Rumbaugh, M. Blaha, W. Prmerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.

[Rum99] J. Rumbaugh, I. Jacobson and G Booch, The Unified Modeling Language Reference Manual. Addison-Wesley, Object Technology Series, 1999.

[Sha96] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, 1996.

[Smi77] J. Smith and D. Smith, “Database Abstractions: Aggregation and Generalization”, ACM Transactions on Database Systems Vol. 2 No. 2, 1977, 105-133.

[Sou93] J. Souquières and N. Levy, “Description of Specification Developments”, *Proc. RE’93 - First IEEE Symposium on Requirements Engineering,* San Diego, 1993, 216-223.

[Sta95] The Standish Group, "Software Chaos", http://www.standishgroup.com/chaos.html.

[Sut97] A. Sutcliffe, “A Technique Combination Approach to Requirements Engineering”, *Proc. RE’97 - 3rd Intl. Symp. on Requirements Engineering,* Anapolis, IEEE, 1997, 65-74.

[Tay96] R. Taylor, N. Medvidovic, K. Anderson J. Whitehead, J. Robbins, K. Nies, P. Oreizy, and D. Dubrow, "A Component- and Message-Based Architectural Style for GUI Software, IEEE Transactions on Software Engineering, Vol. 22 No. 6, June 1996, 390-406.

[Tho99] J.M. Thompson, M.E. Heimdahl, and S.P. Miller, “Specification-Based Prototyping for Embedded Systems”, *Proc. ESEC/FSE’99*, Toulouse, ACM SIGSOFT, LNCS 1687, Springer-Verlag, 1999, 163-179.

[War85] P.T. Ward and S.J. Mellor, *Structured Development for Real-Time Systems*. Prentice-Hall, 1985.

[Was79] A. Wasserman, “A Specification Method for Inteactive Information Systems”, *Proceedings SRS - Specification of Reliable Software*. IEEE Catalog No. 79 CH1401-9C, 1979, 68-79.

[Wei98] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer, “Scenario Usage in System Development: A Report on Current Practice”. *IEEE Software,* March 1998.

[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.

[YuM94] E. Yu and J. Mylopoulos, "Understanding 'why' in software process modeling, analysis, and design", *Proceedings ICSE'94, 16th International Conference on Software Engineering*, Sorrento, IEEE, 1994, 159-168.

[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.

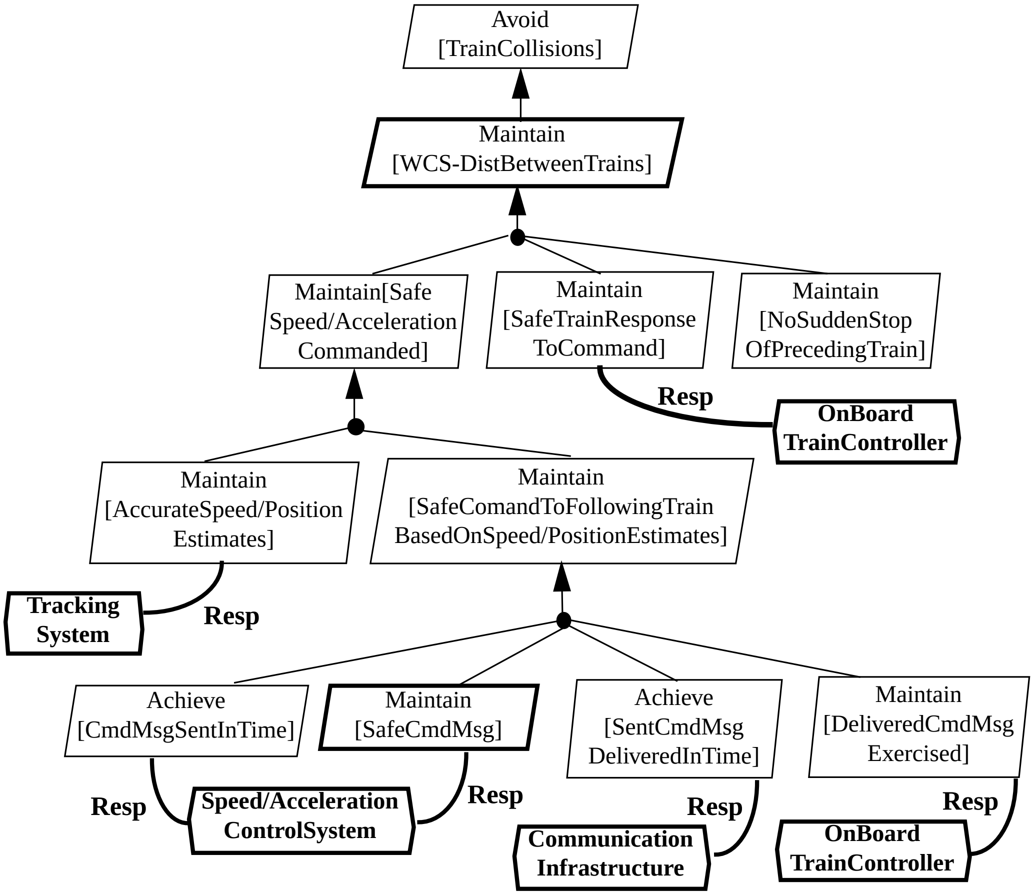
[Zav93] P. Zave and M. Jackson, "Conjunction as Composition", *ACM Transactions on Software Engineering and Methodology*, Vol. 2 No. 4, October 1993, 379-411.

[Zav96] P. Zave and M. Jackson, "Where Do Operations Come From? A Multiparadigm Specification Technique", *IEEE Transactions on Software Engineering*, Vol. 22 No. 7, July 1996, 508-528.

[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


Avoid [TrainCollisions]
Maintain [WCS-DistBetweenTrains]
Maintain[Safe Speed/Acceleration Commanded]
Maintain [SafeTrainResponse ToCommand]
Maintain [NoSuddenStop OfPrecedingTrain]
Resp
OnBoard TrainController
Maintain [AccurateSpeed/Position Estimates]
Maintain [SafeComandToFollowingTrain BasedOnSpeed/PositionEstimates]
Tracking System
Resp
Achieve [CmdMsgSentInTime]
Maintain [SafeCmdMsg]
Achieve [SentCmdMsg DeliveredInTime]
Maintain [DeliveredCmdMsg Exercised]
Resp
Speed/Acceleration ControlSystem
Resp
Resp
Communication Infrastructure
Resp
OnBoard TrainController