From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering

Axel van Lamsweerde and Emmanuel Letier

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

Abstract Requirements engineering (RE) is concerned with the elicitation of the objectives to be achieved by the system envisioned, the operationalization of such objectives into specifications of services and constraints, the assignment of responsibilities for the resulting requirements to agents such as humans, devices and software, and the evolution of such requirements over time and across system families. Getting high-quality requirements is difficult and critical. Recent surveys have confirmed the growing recognition of RE as an area of primary concern in software engineering research and practice.

The paper reviews the important limitations of OO modeling and formal specification technology when applied to this early phase of the software lifecycle. It argues that goals are an essential abstraction for eliciting, elaborating, modeling, specifying, analyzing, verifying, negotiating and documenting robust and conflict-free requirements. A safety injection system for a nuclear power plant is used as a running example to illustrate the key role of goals while engineering requirements for high assurance systems.

Keywords Goal-oriented requirements engineering, high assurance systems, safety, specification building process, lightweight formal methods.

1. INTRODUCTION

The requirements problem has been with us for a long time. An early empirical study over a variety of software projects revealed that inadequate, inconsistent, incomplete, or ambiguous requirements are numerous and have a critical impact on the quality of the resulting software [Bel76]. Late correction of requirements errors was observed to be incredibly expensive [Boe81]. A consensus has been growing that engineering high-quality requirements is difficult; as Brooks noted in his landmark paper on the essence and accidents of software engineering, "*the hardest single part of building a*

In *Radical Innovations of Software & Systems Engineering*, Post-Workshop Proceedings of the Monterey'02 Workshop, Venice (I), Springer-Verlag LNCS, 2003.

software system is deciding precisely what to build" [Bro87]. In spite of such early recognition, the requirements problem is still with us – more than ever. Recent surveys over a wide variety of organizations and projects in the United States and in Europe have confirmed the problem on a much larger scale; poor requirements have consistently been recognized to be the major cause of software problems such as cost overrun, delayed delivery or failure to meet expectations [Sta95, ESI96]. The problem gets even more serious in the case of safety-critical or security-critical systems; most severe failures have been recognized to be traceable back to defective specification of requirements [Lut93, Lev95, Kni02].

Semi-formal modeling notations à la UML and formal specification techniques have been proposed as candidate solutions to address the requirements problem. The strength of the former is their usability (at the price of fairly imprecise semantics), their support for multiple system views and their standardization. The strength of the latter is the wide variety of analysis tools they provide for algorithmic model checking, deductive verification, specification animation, specification-based testing, specification reuse and specification refinement; as a result, the number of success stories in using formal specification technology for real systems is steadily growing from year to year [Lam00c].

In spite of such good news, traditional semi-formal modeling and formal specification techniques suffer from serious weaknesses that explain why they are not fully adequate for the upstream, critical phase of *requirements* elaboration and analysis.

- ***Limited scope***. The vast majority of techniques focus on the modeling and specification of the software alone. They lack support for reasoning about the *composite system* made of the software and its environment. Inadequate assumptions about the environment in which the software operates are however known to be responsible for many errors in requirements specifications [Jac95, Lev95]. *Non-functional requirements* are also generally left outside any kind of treatment. Such requirements form an important part of any specification; they are known to play a prominent role in the evaluation of alternatives, the management of conflicts, the derivation of architectures and evolution management [Chu00, Lam00b, Lam03].
- ***Lack of rationale capture***. Detailed requirements specifications are difficult to understand. Efforts have been made towards formal notations that are more readable [Har87, Heim96, Heit96]. Such efforts however do not address the problem of understanding requirements in terms of their rationale with respect to some higher-level concerns in the application domain.
- ***Poor guidance***. The main emphasis in modeling and specification has been on suitable sets of notations and tools for *a posteriori* analysis. Constructive methods for building correct models/specifications for complex systems in a systematic, incremental way are by and large non-existent. The problem is not merely one of translating natural language statements into some semi-formal model and/or formal specification. Requirements engineering in general requires complex requirements to be elicited, elaborated, structured, interrelated and negotiated.
- ***Lack of support for exploration of alternatives.*** Requirements engineering is much concerned with the exploration of alternative system proposals in which

more or less functionality is automated. Different assignment of responsibilities among software/environment components yield different software-environment boundaries and interactions. Traditional modeling and specification techniques do not allow such alternatives to be represented, explored, and compared for selection.

In this paper, we argue that *goals* offer the right kind of abstraction to address such inadequacies, notably, in the specific context of high assurance systems, that is, systems for which compelling evidence is required that the system delivers its services in a manner that satisfies safety, security, fault-tolerance and survivability requirements [Lea95].

G*oals* are declarative statements of intent to be achieved by the system under consideration [Dar93, Lam00b]. The word "system" here refers to the software-to-be together with its environment [Fea87, Fic92]. Goals are formulated in terms of prescriptive assertions (as opposed to descriptive ones) [Zav97]; they may refer to functional or non-functional properties and range from high-level concerns (such as "safe nuclear power plant") to lower-level ones (such as "safety injection overridden when block switch is on and pressure is less than 'Permit'). *Agents* are system components such as humans playing specific roles, devices and software. A *requirement* is a goal whose achievement is under responsibility of a single software agent. An *expectation* is a goal whose achievement is under responsibility of a single environment agent.

Modeling and reasoning about goals is especially important for high assurance systems as some of the system goals correspond to the application-specific safety, security, fault tolerance and survivability properties that need be achieved with high assurance. Positive/negative interactions with the other system goals can be captured in goal models and managed appropriately [Lam98]; exceptional conditions in the environment that may prevent critical goals from being achieved can be identified and resolved to produce more robust requirements [Lam00a]; the goals can be specified precisely and refined incrementally into operational software specifications that provably assure the higher-level goals [Dar96, Let02a, Let02b]. Requirements in fact "implement" goals much the same way as programs implement design specifications.

The paper discusses the relevance and benefits of explicitly modeling and reasoning about goals at various levels of abstraction in the specific context of high assurance systems. We illustrate the use of a comprehensive set of goal-oriented techniques to build and analyze the requirements for a safety injection control system [Cou93]. Although fairly small, this case study comes from a real application, raises many of the issues found in high assurance systems and is frequently used to illustrate other methods such as, e.g., the SCR method [Heit96] and its analysis techniques [Bha99, Jef98, Gar99]. Other illustrations involving first-order formalizations can be found in [Lam00a, Lam00b, Let01].

2. GOAL-ORIENTED RE IN ACTION: ELABORATING REQUIREMENTS FOR A SAFETY INJECTION SYSTEM

We follow our KAOS method to gradually derive operational requirements for the safety injection software from the underlying system goals. (KAOS stands for "Keep All Objectives Satisfied").

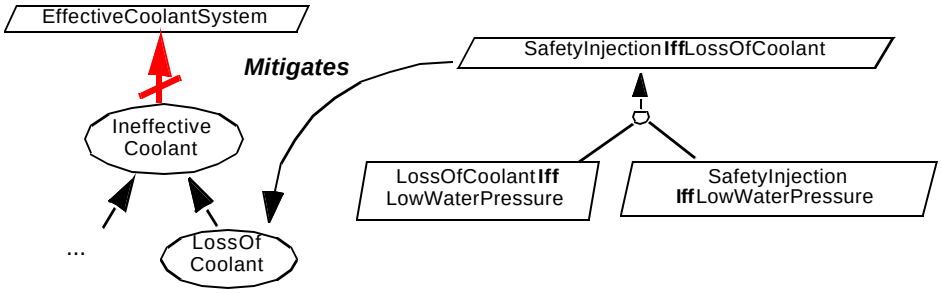

Figure 1: Preliminary goals identified from initial description of the safety injection system [Cou93]

A goal refinement graph is elaborated first by identifying relevant goals from the preliminary system description [Cou93], typically by looking for intentional keywords in natural language statements and by asking *why* and *how* questions about such statements (*goal elaboration step*); conceptual classes, attributes and associations are derived from the goal specification (*object modeling step*); agents are identified together with their potential monitoring/control capabilities, and alternative assignments of goals to agents are explored (*agent modeling step*); operations and their domain pre- and postconditions are identified from the goal specifications, and strengthened pre-, post- and trigger conditions are derived so as to ensure the corresponding goals (*operationalization step*). In parallel, two other steps of the method handle conflicting goals and obstacles that may obstruct goal satisfaction, respectively. The suggested ordering among steps corresponds to an idealized process; in practice however there is significant intertwining and backtracking between them.

Our presentation will be succinct and fragmentary for space reasons; the interested reader may refer to [Let02c] for a full treatment of the case study.

2.1. Goal identification from the source document

Fig. 1 shows some preliminary goals that have been directly identified from the first two paragraphs of the preliminary description of the safety injection system [Cou93]. This figure can be read as follows. One goal in a nuclear power plant is to maintain an effective coolant system (EffectiveCoolantSystem). This goal can be obstructed by an *obstacle* such as LossOfCoolant. (Obstacles may be seen as a high-level faults derived from goal negations; techniques for systematically identifying ways in which a system may fail will be discussed more precisely below.)

The goal SafetyInjection **Iff** LossOfCoolant is introduced to mitigate the obstacle. This goal is then refined into

- an accuracy property about the environment: LossOfCoolant **Iff** LowWaterPressure,
- the subgoal SafetyInjection **Iff** LowWaterPressure.

2.2. Formalizing goals, modeling objects and identifying state variables

Formal analysis techniques may complement informal or semi-formal ones in order to provide higher assurance in the correctness and completeness of the system requirements. Goals then need to be formalized to enable their use. As we will see, goal formalization also allows for more systematic guidance in the requirements elaboration process.

In addition to the usual logical connectives, the following linear temporal operators will be used in this paper:

$\Diamond$ P	P holds in some future state
$\Box$ P	P holds in all future states
A $\Rightarrow$ C	In every future state A implies C, i.e., $\Box$(A $\rightarrow$ C)
A $\Leftrightarrow$ C	In every future state A is equivalent to C, i.e., $\Box$ (A $\leftrightarrow$ C)
$\bullet$ P	P holds in the previous state
@ P	P has just become true, i.e., $\bullet\neg$ P $\wedge$ P

For example, the goal Maintain[SafetyInjection**Iff**LowWaterPressure] may be defined as follows:

Goal Maintain [SafetyInjection **Iff** LowWaterPressure]

InformalDef *The safety injection signal should be 'On' when and only when the water pressure is below the 'Low' set point.*

FormalDef SafetyInjectionSignal = 'On' $\Leftrightarrow$ WaterPressure < 'Low'

The above goal refers to state variables WaterPressure and SafetyInjectionSignal that are declared as attributes of corresponding conceptual classes in a preliminary object model (see Fig. 2).

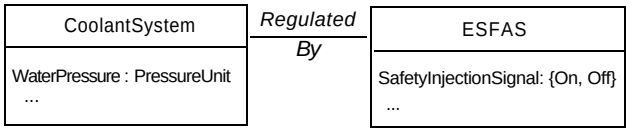

Figure 2: Goal-driven object modeling

These attributes receive the following physical interpretation:

WaterPressure: *the actual pressure of water in the coolant system*

SafetyInjectionSignal: *signal sent by the ESFAS (Engineered Safety Feature Actuation System) to safety features components to command the actual safety injection mechanisms*

Conceptual classes, attributes and associations are incrementally identified and defined as the requirements model is elaborated. When first-order formalizations are used, associations are typically derived from atomic formulas involved in the formal goal assertions [Lam00b]. As opposed to standard OO modeling where it is never clear how and why such or such class/attribute/association should enter the picture, *goal-based object modeling is grounded on a precise criterion for identifying elements of the object model*; such elements are modelled when and only when they are involved in declarative assertions about goals and requirements. Also note the difference with use-case driven modeling; here we start from higher-level, general, declarative and precise statements of intent rather than generally overspecific, operational and often imprecise

descriptions of operations achieving goals left implicit. In fact, use cases can be trivially generated at the very last, operationalization step of our method (see Section 2.7).

2.3. Detecting and resolving goal-level conflicts

Another goal appearing in the available source document is to avoid actuation of the safety injection system during normal start-up or cool down phases:

Goal Avoid [SafetyInjectionDuringNormalStartUp/CoolDown]

InformalDef *Safety injection signals should not be sent during normal start-up or cool down.*

FormalDef (NormalStartUp $\vee$ NormalCoolDown) $\Rightarrow$ SafetyInjectionSignal = 'Off'

This new goal introduces a conflict with the goal Maintain[SafetyInjection**Iff**LowWaterPressure] previously identified. This conflict is detected formally using a predefined conflict pattern from [Lam98]. The two goals are in fact not logically inconsistent; however, they become inconsistent when the plant is in start-up or cool down phase and the water pressure is below 'Low'. This condition is called *boundary condition for conflict* [Lam98]; its formal definition is generated formally by instantiation of our formal conflict pattern which yields:

$$\Diamond\,(\,(\text{NormalStartUp} \vee \text{NormalCoolDown}) \wedge \text{WaterPressure} < \text{'Low'})$$

Conflict resolution tactics from [Lam98] may then be used to propose alternative resolutions; in this case, the conflict is resolved by *weakening* the goal Maintain[SafetyInjection**Iff**LowWaterPressure] with the predicate appearing in the boundary condition. We thereby obtain:

Goal Maintain [SafetyInjection**Iff**LowWaterPressure**Except**DuringStartUp/CoolDown]

InformalDef *The safety injection signal should be 'On' whenever there is a loss of coolant, except during normal start-up or cool down.*

FormalDef SafetyInjectionSignal = 'On' $\Leftrightarrow$

$$\text{WaterPressure} < \text{'Low'} \wedge \neg\,(\textit{NormalStartUp} \vee \textit{NormalCoolDown})$$

This goal will be refined and operationalized in the following sections.

2.4. Refining goals and identifying agent responsibilities

Goals have to be refined until they can be assigned as responsibilities of single agents. However, a goal can be assigned to an agent only if this agent has sufficient monitoring and control capabilities to realize the goal [Let02a]. (Our terminology here is based on the 4-variable model [Par95] and the notion of shared phenomena [Jac95].)

For example, the goal Maintain[SafetyInjection**Iff**LowWaterPressure**Except**DuringStartUp/CoolDown] is unrealizable by the 'Engineered Safety Feature Actuation System' (ESFAS) because this agent cannot monitor whether the plant is in normal startup or cooldown phase.

A catalog of agent-based refinement tactics has been defined to guide the process of refining unrealizable goals until realizable subgoals are reached [Let02a]. Each tactic suggests the application of a formal refinement pattern (see Fig. 3).

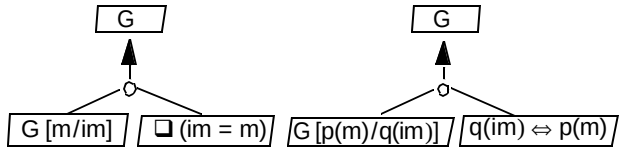

Figure 3: The ' Introduce accuracy goal' tactics

The first tactic in Fig. 3 may be used to resolve ESFAS' lack of monitorability of state variables NormalStartUp and NormalCoolDown. Applying the corresponding pattern yields a new, monitorable state variable, Overridden say, and a refinement of the unrealizable goal Maintain[SafetyInjection**Iff**LowWaterPressure**Except**DuringStartUp/CoolDown] into two subgoals:

- a subgoal SafetyInjection**Iff**LowWaterPressure**Except**When*Overriden*, formally defined by

  SafetyInjectionSignal = 'On' ⇔
  WaterPressure < 'Low' ∧ ¬ *Overridden*

- a companion *accuracy* goal SafetyInjection*Overridden* DuringStartUp/CoolDown, formally defined by

  Overridden ⇔ (NormalStartUp ∨ NormalCoolDown)

Such formal definitions are generated by instantiation of the formal refinement pattern associated with the selected tactic. Goal refinement patterns are proved correct once for all [Dar96]; the STEP verification system [Man96] may be used to check that the conjunction of leaf nodes entails the parent node. At every pattern application the user gets an instantiated proof of correctness of the refinement for free.

The above first subgoal SafetyInjection**Iff**LowWaterPressure**Except**When*Overriden* is now realizable by the ESFAS software agent because it is entirely defined in terms of variables that turn to be monitorable and controllable by this agent; the first subgoal therefore becomes a *requirement* on that agent.

The accuracy subgoal SafetyInjection*Overridden*DuringStartUp/CoolDown is still not realizable by the ESFAS agent because this agent still lacks monitorability of state variables NormalStartUp and NormalCoolDown. Agent-based refinement tactics may again be used to guide the generation of alternative refinements for this goal. One alternative consists in:

(1) introducing two new variables, Block and Reset, that represent manual *block* and *reset* buttons controlled by a human Operator agent;

(2) assigning to the Operator agent the responsibility of pushing the *block* button when and only when the plant enters normal cooldown/startup, and the responsibility of pushing the *reset* button when and only when the plant leaves normal cooldown/startup (the latter two subgoals turn out to be realizable by the Operator agent and therefore become environment *assumptions*); and

(3) assigning to the ESFAS agent the responsibility of overriding safety injection if and

only if 'block' is pushed, and the responsibility of enabling safety injection if and only if 'reset' is pushed (the latter two subgoals turn out to be realizable by the ESFAS software agent and therefore become *software requirements*).

Further details about the generated goal graph and responsibility assignments may be found in [Let02c].

Note that both *software requirements and environmental assumptions are in general needed to prove higher-level goals*.

2.5. Deriving agent interfaces

Capturing the agents' monitoring and control capabilities is an important aspect of the requirements elaboration process [Fea87, Par95, Jac95]. Such capabilities were gradually identified during the previous goal refinement step. The resulting agent interface model for the safety injection system is shown in Fig. 4. It corresponds to a context diagram [Jac95].

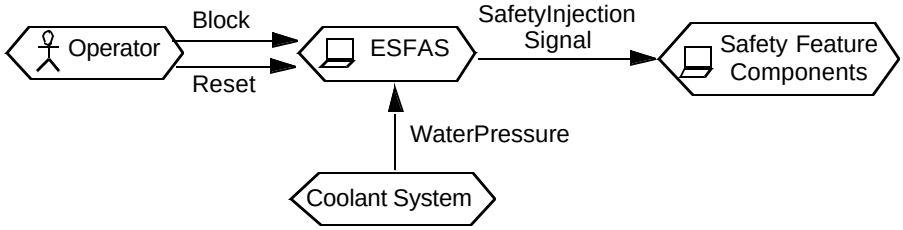

Figure 4: Derived agent interface model for the safety injection system

Note that *alternative goal refinements and alternative responsibility assignments in general lead to alternative software-environment boundaries*, that is, *alternative system proposals and agent interfaces in which more or less is automated*.

2.6. Generating and resolving obstacles to goal achievement

First-sketch specifications of goals, requirements and assumptions tend to be over-ideal; 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. We capture such exceptional behaviors by formal assertions called *obstacles* to goal satisfaction.

An obstacle *O* is said to obstruct a goal *G* iff

$$\{O, Dom\} \models \neg G \qquad \textit{obstruction}$$
$$Dom \not\models \neg O \qquad \textit{domain consistency}$$

Obstacle analysis consists in taking a pessimistic view at the goals, requirements, and assumptions elaborated. The idea is to identify as many ways of breaking such properties as possible in order to resolve them and produce more complete requirements for more robust systems.

We just illustrate a few results from obstacle analysis for some of the terminal goals we identified before. For example, in the previous goal refinement process, we made

the following idealized assumption on the behavior of the Operator agent:

Assumption Avoid[ManualBlock **When** NoStartUp/CoolDown]

InformalDef *The block button should not be pushed when the plant is not entering normal startup or cool down.*

FormalDef $\neg$ @ (NormalStartUp $\vee$ NormalCoolDown) $\Rightarrow$ $\neg$ @ (Block = 'On')

UnderResponsibilityOf Operator

In this case, by just taking the negation of the above assumption we would identify the following obstacle:

Obstacle OperatorPushesBlock **WhenNot** InStartUp/CoolDown

InformalDef *'Block' is pushed when the plant is not entering normal startup or cool down.*

FormalDef $\Diamond$ ($\neg$ @ (NormalStartUp $\vee$ NormalCoolDown) $\wedge$ @ (Block = 'On'))

Similarly, from the assumption Achieve[ManualResetOnExitFromStartUp/CoolDown] assigned to the Operator agent, we would identify the obstacle OperatorForgetsToReset. Other obstacles to assumptions on the Operator agent and to requirements on the ESFAS agent can be identified in the same way [Let02c].

Formal techniques for obstacle generation and refinement are detailed in [Lam00a]. The basic technique amounts to a precondition calculus that regresses goal negations backwards through known properties about the domain; formal obstruction patterns may be used as an alternative to shortcut formal derivations. A formal completeness criterion is also given in [Lam00a]; such completeness is bound by the set of properties known about the domain. Our techniques allow the analyst to incrementally elicit new domain properties as well.

Obstacles should be resolved once they have been generated. Obstacle resolution involves assessing the likelihood and criticality of the obstacle, investigating alternative ways of resolving it, and choosing one resolution alternative based on various criteria such as cost, risks, performance, etc.

Obstacle resolution tactics may be used to generate alternative resolutions [Lam00a]. For example, one of our tactics yields a resolution of the obstacle OperatorPushesBlock-WhenNotInStartUp/CoolDown in which an alternative refinement of the higher-level goal SafetyInjectionOverriddenDuringStartUp/CoolDown is considered; in this alternative, the responsibility of the Operator agent is *weakened*, so as to partially cover the obstacle, whereas the responsibility of the ESFAS agent is *strengthened*. Such an alternative design might be identified by observing that pushing the block button when the water pressure is above some specified value 'Permit' is necessarily an Operator's error because of a domain property stating that the plant cannot be in normal startup/cooldown at such high pressure. Accordingly, the requirement on the ESFAS agent is strengthened so that safety injection does *not* become overridden if the block button is pushed when the water pressure is above 'Permit':

Goal Maintain [SafetyInjectionOverridden **When** BlockSwitchOn **And** *PressureLessThanPermit*]

InformalDef *Safety injection should become overridden when, and only when, the block switch is set to 'On' while the water pressure is less than 'Permit'.*

FormalDef @ Overridden $\Leftrightarrow$

@ (Block = 'On') $\wedge$ *WaterPressure* $\leq$ *'Permit'* $\wedge$ $\bullet$ $\neg$ Overridden

UnderResponsibilityOf ESFAS

The obstacle OperatorForgetsToReset is resolved in a similar way by weakening the responsibility of the Operator agent and strengthening the responsibility of the ESFAS agent. In this case, the requirement of the ESFAS agent is strengthened so that safety injection becomes automatically enabled when the water pressure raises above 'Permit'.

Our resolution tactics so far include goal substitution, agent substitution, goal weakening, goal restoration, obstacle prevention and obstacle mitigation [Lam00a]. In general several generated resolutions will be applicable so that a "best" alternative needs to selected according to non-functional goals from the goal graph (we come back to this below). The selection and application of a resolution may be carried out at specification time, to produce more robust requirements specifications, or at run time, when a requirements monitor detects that the obstacle does occur or is likely to occur [Fea98].

Note that obstacle analysis is an iterative process; it may produce new goals for which new obstacles may need to be identified. In the resulting software specification, some of the obstacles may be totally or partially resolved, some obstacles may remain unchanged (e.g., if they are highly unlikely, do not matter or are deferred to run time) and some new obstacles may appear as a result of previous resolutions.

As mentioned before, the selection among alternative resolutions and the decision to iterate further obstacle analysis cycles should be based on some trade-off assessment among various non-functional, application-specific goals about safety, security, cost, performance, etc. This is an area where much work remains to be done. Qualitative techniques might help here by exposing the competing influences of various alternatives with respect to non-functional goals. A preliminary proposal can be found in [Chu00] where a procedure is proposed for propagating positive/negative influences along alternative paths in the goal graph. For high assurance systems, however, more accurate, quantitative techniques are required. For example, probabilistic risk assessment techniques might provide more precise input to the decision making process. Such techniques, however, rely on the availability of accurate estimates of probabilities of failure events. Obtaining such data may be problematic; the use of such quantitative techniques has therefore been controversial [Lev95]. The real challenge is probably to define a decision process that combines *qualitative reasoning* for those non-functional aspects of the system for which no accurate quantitative weighting can be made and *quantitative reasoning* for those non-functional aspects for which meaningful weighting can be obtained.

Obstacle analysis may be seen as a goal-oriented, formal, constructive method for building fault trees and recovery actions. It is particularly relevant to high assurance systems as many problems and failures of such systems are known to be caused by poor designs that are unable to cope with errors caused by humans, devices and software [Lev95].

2.7. Deriving operational requirements from system goals

The next step of the requirements elaboration process consists in deriving operational software specifications from the terminal goals assigned to software agents. The result is an operation model that defines the various services to be provided by the software in terms of their pre-/postcondition in the domain and strengthened conditions ensuring that the underlying goals in the goal model are met by the services.

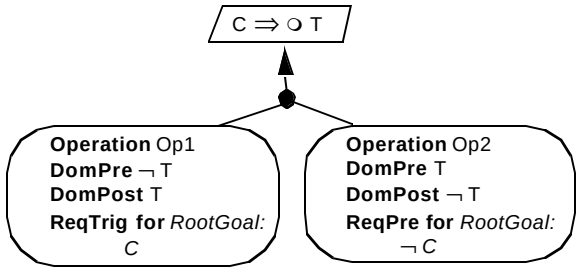

Figure 5: The 'Immediate Achieve' pattern

A catalog of formal operationalization patterns is available to support the operationalization step [Let02b]. For example, the 'Immediate Achieve' pattern is shown in Fig. 5.

Let us come back to the goal

Maintain [SafetyInjectionOverridden**When**BlockSwitchOn**And***PressureLessThanPermit*]

that we assigned to the ESFAS agent, and to the right-to-left implication in the formal definition of this goal given in Section 2.6. The 'Immediate Achieve' operationalization pattern can be used to derive the following operational requirements:

Operation OverrideSafetyInjection
PerformedBy ESFAS
Input Block, WaterPressure; **Output** Overridden
DomPre ¬ Overridden
DomPost Overridden
ReqPre/TrigFor SafetyInjectionOverridden**When**BlockSwitchOn
AndPressureLessThanPermit:
@ (Block = 'On') ∧ WaterPressure ≤ 'Permit'

Note that a distinction is made between *domain* pre- and postconditions that capture what any application of the operation means in the application domain, and *required* pre-, trigger, and postconditions that capture requirements on the operations that are necessary to achieve the goals. (Such distinction somewhat corresponds to the distinction between indicative and optative properties in [Jac95, Zav97].)

In the above operation, the ReqPre/Trigger keyword is a syntactic shortcut to express that the condition is both a required pre- *and* a required trigger- condition for the satisfaction of the corresponding goal; the operation *may* be applied *only if* the condition is true and *must* be applied *if* the condition becomes true and the domain precondition is true.

Similarly, from the goal Maintain[SafetyInjectionEnabled**When**PressureAbovePermit**Or**ManualReset], we can systematically derive the need for an operation EnableSafetyInjection together with strengthened conditions on this operation that will guarantee the satisfaction of this goal. Specifications for the operations SendSafetyInjectionSignal and StopSafetyInjectionSignal are similarly derived from the specification of the goal Maintain[SafetyInjection**When**LowWaterPressure**AndNot**Overridden], see [Let02c] for details.

It is worth noting that our *goal-oriented requirements elaboration process ends where most traditional specification techniques would start.* For example, the operational

specifications obtained above can be mapped to SCR tables for the same system through a series of transformation steps each of which resolves a semantic, structural or syntactic difference between the source (KAOS) specification and the target (SCR) one [Del03].

Note again the difference with use-case driven modeling; we started from higher-level, general, declarative and precise statements of intent rather than generally overspecific, operational and often imprecise descriptions of operations achieving goals left implicit. Use cases emerge at the last step of our method as aggregations of the operations that operationalize functional goals assigned to software agents.

3. CONCLUSION

We used a safety injection system as a running example to illustrate the benefits of a constructive, goal-oriented approach to requirements elaboration and analysis. The key points illustrated by this elaboration process are the following.

- Goal-oriented modeling and specification takes a wider system engineering perspective; goals are prescriptive assertions that should hold in the system made of the software-to-be *and its environment*; domain properties and expectations about the environment are explicitly captured during the requirements elaboration process, in addition to the usual software requirements specifications.
- Operational requirements are derived incrementally from the higher-level system goals they "implement".
- Goals provide the rationale for the requirements that operationalize them and, in addition, a correctness criterion for requirements completeness and pertinence [Yue87].
- Obstacle analysis helps producing much more robust systems by systematically generating (a) potential ways in which the system might fail to meet its goals and (b) alternative ways of resolving such problems early enough during the requirements elaboration and negotiation phase.
- Alternative system proposals are explored through alternative goal refinements, responsibility assignments, obstacle resolutions and conflict resolutions.
- The goal refinement structure provides a rich way of structuring and documenting the entire requirements document.
- A multiparadigm, 'multi-button' framework allows one to combine different levels of expression and reasoning: *semi-formal* for modeling and structuring, *qualitative* for selection among alternatives, and *formal*, when needed, for more accurate reasoning.
- Goal formalization allows RE-specific types of analysis to be carried out, such as
  - guiding the goal refinement process and the systematic identification of objects and agents [Lam00b, Let02a];
  - checking the correctness of goal refinements and detecting missing goals and implicit assumptions [Dar96];
  - guiding the identification of obstacles and their resolutions [Lam00a];

- guiding the identification of conflicts and their resolutions[Lam98];
- guiding the identification and specification of operational requirements that satisfy the goals [Dar93, Let02b].

Several important topics are however not yet sufficiently addressed by current goal-oriented techniques.

- Current support for the evaluation and selection among multiple alternatives explored during the requirements elaboration process is highly limited. As discussed before, a blend of qualitative and quantitative reasoning techniques should be devised for more accurate evaluation of alternatives in terms of measurable quantities. Such techniques should probably be based on specific models for specific types of non-functional goals, e.g., risk models for safety goals, cost models for cost-related goals, performance models for performance-related goals, etc.
- Much work also remains to be done to provide *specialized* techniques for goal refinement and obstacle/conflict analysis that are targeted to *specific goal categories* relevant to high assurance systems (e.g., safety or security) and to specific domains (e.g., air traffic control, medical applications). This means characterizing and refining goal categories more thoroughly (maybe in domain-specific terms at some point), defining suitable notations and techniques for modeling and specifying properties in each category, and finding systematic ways of reasoning about their positive/negative interactions *at the goal level.*
- Further work is also needed to integrate the methodological support provided by our goal-oriented requirements engineering method with existing specification analysis tools. Such integration may occur at two levels. First, we would like to use existing tools to automate some of the *RE-specific* formal reasoning described above. For example, we recently built a tool prototype for early model checking of *goal* models; the generated counter-examples suggest inconsistent or missing goals. Second, we would like to map the result of goal-oriented requirements elaborations to specialized tools for formal analysis of operational specifications. For example, we did a mapping of KAOS models to SCR tables [Del03]. Other mappings, e.g., to the NuSMV model checker (http://nusmv.irst.itc.it/) or the Alloy analyzer (http://sdg.lcs.mit.edu/alloy/), are under way.

Acknowledgement

The work of Emmanuel Letier was supported by the "Fonds National de la Recherche Scientifique" (FNRS). We are grateful to the KAOS/GRAIL crew at CEDITI for using some of the techniques presented here in industrial projects and to members of the FAUST project at CETIC for developing the (much needed) formal analysis toolkit.

REFERENCES

[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.

[Bha99] R. Bharadwaj and C. Heitmeyer, "Model Checking Complete Requirements Specifications Using Abstraction," *Automated Software Engineering*, Vol 6, No. 1, January 1999, 37-68.

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

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

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

[Cou93] P.J. Courtois and D.L. Parnas, "Documentation for Safety-Critical Software", *Proc. ICSE'1993: 15th International Conference on Software Engineering*, ACM Press, 1993, 315-323.

[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.

[Del03] R. De Landtsheer, E. Letier and A. van Lamsweerde, "Deriving Tabular Event-Based Specifications from Goal-Oriented Requirements Models", *Proc. RE'03 - International Joint Conference on Requirements Engineering*, Monterey (CA), IEEE, September 2003. Expanded version to appear in the *Requirements Engineering Journal*.

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

[Fea87] M. Feather, "Language Support for the Specification and Development of Composite Systems", *ACM Trans. on Programming Languages and Systems* 9(2), April 1987, 198-234.

[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.

[Gar99] A. Gargantini and C. Heitmeyer, "Using Model Checking to Generate Tests from Requirements Specifications", *Proc., ESEC'99 & 7th ACM SIGSOFT Intern. Symp. on Foundations of Software Eng. (ESEC/FSE99)*, Toulouse, September 1999.

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

[Heim96] M. Heimdahl and N.G. Leveson, "Completness and Consistency Checking 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 Specifications", *ACM Transactions on Software Engineering and Methodology*, Vol. 5, No. 3, July 1996, 231-261.

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

[Jef98] R. Jeffords and C. Heitmeyer, "Automatic Generation of State Invariants from Requirements Specifications", *6th International Symposium on the Foundations of Software Engineering (FSE-6)*, Orlando FL, November 1998.

[Kni02] J.C. Knight, "Safety-Critical Systems: Challenges and Directions", Invited Mini-Tutorial, *Proc. ICSE'2002: 24th International Conference on Software Engineering*, ACM Press, 2002, 547-550.

[Lam98] A. van Lamsweerde, R. Darimont and E. Letier, "Managing Conflicts in Goal-driven Requirements Engineering", *IEEE Transactions on Software Engineering*, Special Issue on Inconsistency Management in Software Development, Vol. 24, No. 11, November 1998, 908-926.

[Lam00a] A. van Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering", *IEEE Transactions on Software Engineering*, Special Issue on Exception Handling, Vol. 26, No. 10, October 2000, 978-1005.

[Lam00b] 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, 5-19.

[Lam00c] A. van Lamsweerde, "Formal Specification: a Roadmap", in *The Future of Software Engineering*, A. Finkelstein (ed.), ACM Press, 2000.

[Lam03] A. van Lamsweerde, "From System Goals to Software Architecture", In *Formal Methods for Software Architecture*, M. Bernardo & P. Inverardi (eds.), LNCS 2804, Springer-Verlag, 2003.

[Lea95] J. McLean and C. Heitmeyer, "High Assurance Computer Systems: A Research Agenda", America in the Age of Information, National Science and Technology Council Committee on Information and Communications Forum, Bethesda, 1995.

[Let01] E. Letier, *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD Thesis, Université Catholique de Louvain, Dépt. Ingénierie Informatique, Louvain-la-Neuve, Belgium, May 2001. http://www.info.ucl.ac.be/people/eletier/thesis.html

[Let02a] E. Letier and A. van Lamsweerde, "Agent-Based Tactics for Goal-Oriented Requirements Elaboration", *Proc. ICSE'02: 24th Intl. Conf. on Software Engineering*, Orlando, IEEE Computer Society Press, May 2002.

[Let02b] E. Letier and A. van Lamsweerde, "Deriving Operational Software Specifications from System Goals", *Proc. FSE'10:* 10th *ACM SIGSOFT Symp. on the Foundations of Software Engineering*, Charleston, November 2002.

[Let02c] E. Letier, *Goal-Oriented Elaboration of Requirements for a Safety Injection Control System*. Research Report, Département d'Ingénierie Informatique, UCL, June 2002.

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

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

[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, pp. 415-418.

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

[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.

[Zav97] P. Zave and M. Jackson, "Four dark corners of requirements engineering", *ACM Trans. on Software Engineering and Methodology*, Vol. 6, No. 1, January 1997, 1 - 30.