Integrating Exception Handling in Goal Models

Antoine Cailliau and Axel van Lamsweerde
ICTEAM – Institute for Information & Communication Technologies, Electronics and Applied Mathematics
Université catholique de Louvain
Louvain-la-Neuve, Belgium
{antoine.cailliau, axel.vanlamsweerde}@uclouvain.be

Abstract—Missing requirements are known to be among the major sources of software failure. Incompleteness often results from poor anticipation of what could go wrong with an over-ideal system. Obstacle analysis is a model-based, goal-anchored form of risk analysis aimed at identifying, assessing and resolving exceptional conditions that may obstruct the behavioral goals of the target system. The obstacle resolution step is obviously crucial as it should result in more adequate and more complete requirements. In contrast with obstacle identification and assessment, however, this step has little support beyond a palette of resolution operators encoding tactics for producing isolated countermeasures to single risks. In particular, there is no single clue to date as to where and how such countermeasures should be integrated within a more robust goal model.

To address this problem, the paper describes a systematic technique for integrating obstacle resolutions as countermeasure goals into goal models. The technique is shown to guarantee progress towards a complete goal model; it preserves the correctness of refinements in the overall model; and keeps the original, ideal model visible to avoid cluttering the latter with a combinatorial blow-up of exceptional cases. To allow for this, the goal specification language is slightly extended in order to capture exceptions to goals separately and distinguish normal situations from exceptional ones. The proposed technique is evaluated on a non-trivial ambulance dispatching system.

Index Terms—Obstacle analysis, goal modeling, probabilistic goals, risk control, requirements completeness, exception handling, goal-oriented requirements engineering, quantitative reasoning.

I. INTRODUCTION

Requirements-related errors are commonly recognized to be the most frequent, persistent, expensive and dangerous types of software errors [13]. Among these, missing requirements tend to be the worst. They often arise from a natural inclination to believe that the software and its environment will always behave as expected; no requirements are engineered for cases where this optimistic assumption does not hold. Requirements completeness therefore calls for putting risk analysis at the heart of the RE process [2, 3, 5, 8, 13, 14, 20].

A risk is an uncertain factor whose occurrence may result in the loss of satisfaction of some high-level objective [4, 8, 13]. A risk has a probability of occurrence and one or multiple consequences. Each consequence has a severity in degree of loss of satisfaction of the corresponding objective [5, 8]. Risks may cover undesirable situations such as safety hazards [16, 18], security threats [12, 26] or data inaccuracies [14] dependent on the type of objective they negatively impact on.

At requirements engineering time, risks should be identified, assessed in terms of their likelihood and criticality, and controlled through effective countermeasures [13]. Obstacle analysis has been introduced and used as a model-based, goal-oriented form of risk analysis [2, 6, 14, 21]. An obstacle to a goal is a precondition for non-satisfaction of this goal. Obstacle analysis consists of (a) identifying obstacles from available goals, assumptions and domain properties; (b) assessing their likelihood and criticality in terms of severity of their consequences; and (c) resolving likely and critical obstacles through countermeasures to be incorporated into the goal model.

To support obstacle analysis, techniques are available for identifying obstacles systematically from goals and domain properties [1, 14]. For obstacle assessment, likelihoods and criticalities may be determined qualitatively by calculations over obstacle refinement trees and goal refinement trees, respectively; such calculations call for probabilistic extensions to cope with probabilistic goals and obstacles [5, 25]. For obstacle resolution, operators encoding risk control tactics were proposed to explore alternative resolutions—such as avoid obstacle, reduce obstacle likelihood, mitigate obstacle, weaken goal, substitute goal, restore goal, or substitute agent [14].

The obstacle resolution step is obviously crucial; it directly impacts the adequacy, completeness and robustness of the goal model. However, little support is currently available for this step beyond the above resolution operators for countermeasure exploration. In particular, it is totally unclear where and how selected countermeasures produced by such operators should be integrated in the goal model to increase its completeness and robustness.

To address this problem, the paper describes techniques for integrating obstacle resolutions systematically in the goal refinement graph while propagating the resulting changes whenever required in the model. These techniques guarantee that:

- the model is increasingly robust and complete as resolutions are being integrated;
- the normal system behaviors and those not affected by the obstacles are preserved;
- the correctness of goal refinements in the model is preserved.

A goal model integrating countermeasures to obstacles may need to be restructured so as to keep the goals referring to normal situations separate from the countermeasure goals referring to exceptional situations. There are multiple reasons for this.

- For higher readability and better visibility, the ideal model containing all functional and non-functional goals in normal situations should be kept visible.
specification of these goals should not be cluttered with items referring to exceptional situations.

- The model structuring and specification should not exhibit any combinatorial blow-up of exceptional cases. Without any structuring mechanism, the integration of multiple countermeasures to multiple risks considered in combination might produce a large number of cases.

- Exceptional situations should be identifiable and integrated incrementally. The handling of each single situation should be isolated from the others.

- The traceability of exceptional cases should be supported from requirements to architecture. Keeping exceptions separate from each other and from the goals in normal situations enables traceability from the goal model and its operationalization on the one hand and exception handlers in the architecture on the other hand.

- In case of obstacle tolerance with no countermeasure integrated in the model, a one-to-one mapping should be maintained between (a) the obstacle and countermeasure identified at modeling time, and (b) the corresponding runtime monitor/adaptor mechanisms for dynamic reconfiguration when the obstacle is too frequent [9].

To support such separation between normal and exceptional situations, the paper extends the goal language with semantics-preserving constructs for specifying exceptions and their “handlers” — that is, the countermeasures associated with them. Model transformation operators are then provided for attaching and detaching exceptions to/from associated goals in the goal model.

The paper is organized as follows. Section II introduces some necessary background on modeling goals and their obstacles. Section III motivates our technique on a small example. Section IV more precisely specifies the problem to be solved by our integration approach. Section V describes the conditions and mechanisms for integrating countermeasures in a goal model. Section VI presents the constructs for specifying goals with exceptions together with their semantics. Section VII introduces model transformation operators for attaching and detaching exceptions to/from goals. Section VIII summarizes the evaluation of our proposals on an ambulance dispatching system. Section IX briefly discusses related work.

![Diagram of a partial goal model for a mine pump system](image)

Fig. 1. Partial goal model for a mine pump system

II. BACKGROUND

This section recalls some basics on behavioral goal modeling, obstacle analysis and probabilistic goals while introducing our running example.

**Goal-oriented system modeling.** A goal is a prescriptive statement to be satisfied by cooperating agents forming the considered system. The latter may include devices such as sensors and actuators, people, preexisting software, and the software to be developed. **Domain properties** are descriptive statements about the problem space, e.g., physical laws.

Behavioral goals prescribe maximal sets of desired system behaviors; unlike soft goals they are satisfiable in a clear-cut sense [13]. A **behavior** is a sequence of system state transitions. Linear Temporal Logic (LTL) may be used to specify behavioral goals formally and enable formal analyses [13, 22]. The goals then have the general form:

\[ C \Rightarrow \Theta T, \]

where \( \Theta \) represents an LTL operator such as: \( \circ \) (in the next state), \( \diamond \) (sometimes in the future), \( \diamond_{sd} \) (sometimes in the future before deadline \( d \)), \( \Box \) (always in the future), \( \Box_{sd} \) (always in the future up to deadline \( d \)), \( W \) (always in the future unless), \( U \) (always in the future until), and where \( P \Rightarrow Q \) denotes \( \Box(P \rightarrow Q) \). The following standard logical connectives are used: \( \land \) (and), \( \lor \) (or), \( \neg \) (not), \( \rightarrow \) (implies), \( \leftrightarrow \) (equivalent).

Among behavioral goals, **Achieve** goals follow the specification pattern “if \( C \) then sooner-or-later \( T \)”, that is, \( C \Rightarrow \Theta T \), where \( C \) and \( T \) denote a current and target condition, respectively. **Maintain** goals follow the specification pattern “if \( C \) then always \( G \)”, that is, \( C \Rightarrow \Box G \) where \( G \) denotes a good condition. **Avoid** goals follow a similar pattern “if \( C \) then never \( B \)”, that is, \( C \Rightarrow \Box \neg B \) where \( B \) denotes a bad condition.

A **goal model** is an AND/OR graph showing how goals contribute positively or negatively to each other. Parent goals are obtained by abstraction, e.g., through **why** questions. Subgoals are obtained by refinement, e.g., through **how** questions. In a goal refinement graph, leaf goals are **requirements or assumptions** dependent on whether they are assigned to single software-to-be or environment agents, respectively.

Fig. 1 shows a partial goal model for a mine pump system [11, 13, 15]. Refinement patterns help building such a model through common goal decomposition tactics such as Milestone-Driven, Case-Driven, Guard-Introduction, Divide-And-Conquer, Uncontrollability-Driven, etc. [7, 13]. For example, consider the following goal in Fig. 1:

**Goal** Maintain \([\text{PumpOnWhenHighWater}]\)

**FormalSpec** \( \forall p: \text{Pump}, s: \text{Sump} \)

\[ s.\text{WaterLevel} = "\text{High}" \land \text{PumpInSump}(p, s) \Rightarrow \text{p.Motor} = "\text{On}" \]

Applying the **Unmonitorability-driven** refinement pattern to this goal yields, after instantiation, the following refinement where the antecedent of the second subgoal becomes monitorable by the software pump controller:

**Goal** Maintain \([\text{HighWaterDetected}]\)

**FormalSpec** \( \forall p: \text{Pump}, s: \text{Sump}, c: \text{PumpController} \)

\[ s.\text{WaterLevel} = "\text{High}" \land \text{PumpInSump}(p, s) \land \text{CtrlPump}(c, p) \Rightarrow c.\text{HighWaterSignal} = "\text{On}" \]

**Goal** Maintain \([\text{PumpOnWhenHighWaterDetected}]\)

**FormalSpec** \( \forall p: \text{Pump}, c: \text{PumpController} \)

\[ c.\text{HighWaterSignal} = "\text{On}" \land \text{CtrlPump}(c, p) \Rightarrow \text{p.Motor} = "\text{On}" \]
Refinement patterns produce goal refinements guaranteed to be complete, consistent and minimal. A refinement is complete when the subgoals \( SG_i \), possibly with domain properties in \( Dom \), are sufficient for satisfying the parent goal \( PG \):

\[
\{SG_1, \ldots, SG_n, Dom\} \models PG
\]

A refinement is consistent if:

\[
\{SG_1, \ldots, SG_n, Dom\} \neq false
\]

A refinement is minimal if all subgoals are needed for satisfaction of the parent goal:

\[
\text{for all } 1 < i < n: \{SG_1, \ldots, SG_{i-1}, SG_{i+1}, \ldots, SG_n, Dom\} \neq PG
\]

The two lower AND-refinements in the upper part of Fig. 1 are complete, consistent and minimal.

**Obstacle analysis.** An obstacle to a goal is a satisfiable precondition for not satisfying this goal [14]:

\[
\{O, Dom\} \models \neg G
\]

Obstacles are fine-grained obstacles whose satisfyability and likelihood can be more easily estimated by experts. A variety of formal techniques, obstruction patterns and heuristic rules are available for systematic obstacle identification [1, 13, 14]. For example, consider the following right leaf goal in Fig. 1:

Goal: Maintain \([\text{PumpOnWhenPumpSwitchOn}]\)

FormalSpec: \( Vp: \text{Pump} \)

\(p.\text{Switch} = "\text{On}" \land p.\text{Motor} = "\text{On}"\)

Its negation yields the following root obstacle:

\[
\{O, Dom\} \models \neg G
\]

**Obstacle: PumpFailure**

FormalSpec: \( \Diamond 3p: \text{Pump} \)

\(p.\text{Switch} = "\text{On}" \land p.\text{Motor} \neq "\text{On}"\)

Consider the following domain property capturing a necessary condition for the target \( p.\text{Motor} \neq "\text{On}"\):

\(p.\text{Failure} = true \rightarrow p.\text{Motor} \neq "\text{On}"\)

Regression of the root obstacle backwards through this domain property generates the following subobstacle:

\[
\{O, Dom\} \models \neg G
\]

**Obstacle: PumpFailure**

FormalSpec: \( \Diamond 3p: \text{Pump} \)

\(p.\text{Switch} = "\text{On}" \land p.\text{Failure} = true\)

A variety of tactics are available for exploring alternative ways of resolving obstacles to a goal —such as avoid obstacle, reduce obstacle likelihood, mitigate obstacle, weaken goal, substitute goal, restore goal, or substitute agent [14]. For example, the obstacle mitigation tactic applied to the preceding obstacle PumpFailure may produce the following countermeasure goal:

Goal: Achieve \([\text{MineEvacuatedWhenPumpFailureAndPumpSwitchOn}]\)

FormalSpec: \(Vm: \text{Miner}, p.\text{Pump} \)

\(p.\text{Switch} = "\text{On}" \land p.\text{Failure} = true \rightarrow 0.110_m, m.\text{Position} = "\text{Out}"\)

The lower part of Fig. 1 shows two OR-refinements for the obstacle PumpNotOnAndPumpSwitchOn.

**Probabilistic goals and obstacles.** Behavioral goals may be satisfied only partially [5]. The **probability of satisfaction** for a goal \( G: C \Rightarrow \Theta T \) is defined as the ratio between (a) the number of possible behaviors satisfying both the goal antecedent \( C \) and consequent \( \Theta T \), and (b) the number of possible behaviors satisfying \( C \). The **estimated probability of satisfaction** (EPS) of a goal is the probability of its satisfaction in view of its possible obstructions by obstacles. For goal \( G \), it is denoted by \( P(G) \). The conditional probability \( P(G|H) \) denotes the probability of satisfaction of \( G \) over all behaviors satisfying \( H \).

The **required degree of satisfaction** (RDS) of a goal is the minimal probability of satisfaction admissible for this goal; it is prescribed by elicited requirements, existing regulations, standards, etc. For goal \( G \), it is denoted by \( RDS(G) \).

The **probability of an obstacle** is defined as the ratio (a) the number of possible behaviors satisfying the obstacle, and (b) the number of possible behaviors. The probability of a root obstacle is computed by up-propagation through the obstacle refinement tree from estimates for leaf obstacles. The result is then up-propagated in turn through the AND/OR goal graph to determine the EPS of root goals. The severity of the consequences of obstacles to \( G \) is then assessed from the difference \( P(G) - RDS(G) \) [5].

## III. Integrating Countermeasure Goals: Motivation

In addition to the PumpFailure obstacle in the previous section, the following other obstacle also results in severe loss of satisfaction of high-level goals:

**Obstacle: PowerSystemOutage**

FormalSpec: \( \Diamond 3p: \text{Pump} \)

\(p.\text{Switch} = "\text{On}" \land p.\text{PowerLine} = "\text{Off}"\)

Resolution tactics might produce, among others, the following countermeasure goals to resolve these obstacles (see Fig. 1):

Achieve \([\text{PowerGeneratorOnWhenPowerSystemOutageAndPumpSwitchOn}]\).

Maintain \([\text{PowerEmergencyPumpOnWhenPumpFailureAndPumpSwitchOn}]\).

Maintain \([\text{ThirdPartyPumpOnWhenHighWaterDetected}]\).

Achieve \([\text{PowerGeneratorOnWhenPowerSystemOutageAndPumpSwitchOn}]\).

The paper aims at providing precise and systematic answers to the following questions.

- Where should these countermeasure goals be integrated into the goal refinement graph? How?
- What parent goals should the countermeasure goals refine? With what other sibling subgoals? Should exceptional conditions such as \( p.\text{Failure} = true \) and \( p.\text{PowerLine} = "\text{Off}" \) pollute goal specifications throughout the entire goal model?

The answers to those questions should support the following objectives.

- **Separation of concerns.** The goals referring to normal situations should be distinguished from those handling obstacle occurrences. Such separation may significantly reduce model complexity and keep the ideal, normal model explicit.
- **Compositionality.** It should be possible to specify, structure, and analyze normal goals and countermeasures to their obstacles in a compositional way. A robust model
A goal $G'$ is a deidealized version of goal $G$ if $G \models G'$. A deidealized version $G'$ of $G$ is acceptable if, for every goal refinement with $G$ as a child, there exists an acceptable deidealized version of its siblings and parents such that the corresponding refinement still meets the completeness, consistency and minimality conditions recalled in Section II.

The ancestor entailment condition can now be formulated as follows:

$$\{CG, G_1', \ldots, G_n', Dom\} \models PG' \quad (\text{ancestor-entailment})$$

for some acceptable deidealized version $PG'$ of ancestor $PG$, where $PG$ is an ancestor of the obstructed goal $G$ and $G_1', \ldots, G_n'$ are acceptable deidealized versions of descendants of $PG$. A countermeasure goal $CG$ against obstacle $O$ is said to be valid if it satisfies the non-obstruction and ancestor-entailment conditions.

For example, the countermeasure goal $\text{Achieve [MineEvacuatedWhen PumpFailureAnd PumpSwitchOn]}$ is valid; the obstacle $\text{PumpFailure}$ does not obstruct it, and this goal together with the deidealized goal $\text{Avoid [OverflowedMineWhen PumpFailureAnd PumpSwitchOn]}$ guarantee the satisfaction of the parent goal $\text{Avoid [MinersDrowning]}$. In this example, the parent goal is not deidealized.

Obstacle resolution tactics such as $\text{avoid obstacle, reduce obstacle likelihood, mitigate obstacle, weaken goal, or substitute goal}$ [14] can be shown to produce valid countermeasures modulo propagations of their effect through the refinement trees in which they are involved (see Section V.C).

Multiple candidate ancestors might be considered for the ancestor-entailment condition. In our example, $\text{Avoid [OverflowedMine]}$ and $\text{Avoid [MinersDrowning]}$ are potential candidates with respect to the goal $\text{Achieve [MineEvacuatedWhen PumpFailureAnd PumpSwitchOn]}$. The nearest candidate to this goal appears preferable for more local model change –that is, the goal $\text{Avoid [OverflowedMine]}$.

The anchor for a countermeasure goal $CG$ is the lowest ancestor goal $PG$ meeting the ancestor-entailment condition. It is the goal through which the countermeasure goal is integrated, as discussed in the next section.

**Theorem (Progress).** For any valid countermeasure goal $CG$, the probability of satisfaction of its anchor $PG$ increases:

$$P(PG') > P(PG),$$

where $PG'$ in $M'$ corresponds to $PG$ in $M$.

This can be proved $ab$ *absurdo*. Assume there is no such increase: $P(PG') \leq P(PG)$. Introducing conditional probabilities, we have:

$$P(PG') = P(O) \times P(PG'|O) + P(\neg O) \times P(PG'|\neg O),$$

$$P(PG) = P(O) \times P(PG|O) + P(\neg O) \times P(PG|\neg O).$$

Given the obstruction of $PG$, we have: $P(PG|O) = 0$. Therefore,

$$P(O) \times P(PG'|O) + P(\neg O) \times P(PG'|\neg O) \leq P(\neg O) \times P(PG|\neg O)$$

Since $PG'$ is a deidealized version of $PG$, we have:

$$P(PG'|\neg O) = P(PG|\neg O).$$

Therefore,

$$P(\neg O) \times P(PG'|\neg O) \geq P(\neg O) \times P(PG|\neg O).$$

As $P(O) \times P(PG'|O) = 0$, our initial assumption gets contradicted.
B. Ensuring minimal changes: integration schemas

A single valid countermeasure goal ensures progress towards a complete model; its integration in the model should also ensure that the minimal change property is met (see Section IV).

Two alternative integration schemas may be used for this, dependent on the obstacle resolution tactic being selected [14]. The first schema removes the obstructed goal; it should be applied when the substitute goal or weaken goal tactic is used for resolving the obstacle. The second integration schema keeps the obstructed goal in the model; it should be applied when the avoid obstacle, reduce obstacle likelihood, or mitigate obstacle tactic is used.

Removing the obstructed goal. Fig. 2 shows a first integration schema expressed as a model rewriting rule. In this first schema, the refinement of anchor goal $AG$, containing at least one obstructed goal, is replaced with a new refinement. The latter contains the countermeasure goal $CG$ to leaf obstacle $LO$ together with all non-obstructed children. (An anchor’s obstructed children are those directly or indirectly obstructed by $LO$.) The anchor $AG$ may need to be deidealized; in this case, $AG'$ replaces $AG$ in the new goal model.

This first integration schema has a precondition for use, namely, the countermeasure goal $CG$ and the non-obstructed children are sufficient for satisfying the anchor goal:

$$\{CG, Children(AG) \setminus ObstructedChildren\} \models AG'.$$

Otherwise, the new refinement would not be complete.

For example, the goal Maintain [ThirdPartyPumpOnWhenHighWaterDetected] is a countermeasure produced through the goal substitution tactic [14]. Its anchor goal is Maintain [PumpOnWhenHighWater]. The following refinement is complete, consistent, and minimal for this anchor goal:

Maintain [PumpOnWhenHighWater]
- Maintain [HighWaterDetectedWhenHighWater]
- Maintain [ThirdPartyPumpOnWhenHighWaterDetected]

We may therefore replace the old refinement with this one.

It is easy to see that this first integration schema meets our minimal change property. Non-obstructed goals are composed from non-obstructed goals in the refinements of $AG$ and its descendants, and in the siblings of $AG$ and its ancestors. The former are kept as is whereas the latter might need to be deidealized through change propagation (see Section V.C). All these goals thus satisfy $G' \models G$.

Keeping the obstructed goal. Fig. 3 shows a second integration schema. In this schema, a new refinement is introduced; it includes a modified version of the obstructed anchor and the countermeasure goal. The obstructed anchor is deidealized for removing the obstruction. The negation of the leaf obstacle is added to form a decomposition by cases.

This second integration schema has a precondition for use as well; the countermeasure goal must be sufficient for satisfying the anchor goal when the obstacle occurs:

$$\{LO, CG, Dom\} \models AG'.$$

For example, the countermeasure goal Achieve [MineEvacuatedWhenPumpFailureAndPumpSwitchOn] entails its anchor Avoid [MinersDrowning] when the obstacle occurs. This second rule then produces the following refinement:

Avoid [OverflowedMine]
- Avoid [OverflowedMineWhenNoPumpFailure]
- Achieve [MineEvacuatedWhenPumpFailureAndPumpSwitchOn]

Note that the anchor goal is not deidealized in this example. The new goal Avoid [OverflowedMineWhenNoPumpFailure] is a deidealization of the anchor goal. The negation of the obstacle is added to the goal antecedent:

Goal Avoid [OverflowedMine]
- FormalSpec \(\forall p: Pump\)
  p.Failure = false \(\rightarrow\) ~OverflowedMine

This second integration schema can be seen to meet our minimal change property as well; the reasoning is similar to the first schema. Only siblings of the anchor goal and its ancestors may need to be deidealized. The other goals were obstructed by the resolved obstacle and are therefore not concerned with the minimal change property.

C. Preserving refinement correctness: change propagation

As introduced before, goal deidealizations and countermeasure integrations may require corresponding changes to be propagated along refinement trees in which the countermeasure goal is involved. Such propagations are intended to ensure our third property on integrations, that is, the resulting model must remain complete, consistent, and minimal. This section discusses how deidealizations and change propagations are performed.

Deidealization by strengthening the goal’s antecedent. A first way of deidealizing a goal of form $C \Rightarrow \Theta T$ is to add an adequate conjunct to its antecedent:

AddConjunct ($C \Rightarrow \Theta T, \Theta EC) = \{C \land \Theta EC\} \Rightarrow \Theta T$.

For example, the goal Maintain [PumpOnWhenPumpSwitchOn] may be deidealized so as to exclude pump failure from pump actuation:

Goal Maintain [PumpOnWhenNoPumpFailureAndPumpSwitchOn]
- FormalSpec \(\forall p: Pump\)

Deidealization by weakening the goal’s consequent. A second way of deidealizing the goal $C \Rightarrow \Theta T$ is to add an adequate disjunct to its consequent:

AddDisjunct ($C \Rightarrow \Theta T, \Theta ED) = C \Rightarrow [\Theta T \lor \Theta ED]$. 
For example, the goal \texttt{Maintain[PumpOnWhenPumpSwitchOn]} might be deidealized so as to require the pump to be actuated or the emergency pump to be activated:

\texttt{Goal \textit{Maintain}[PumpOnOn\textit{OrEmergencyPumpOnWhenPumpSwitchOn}]}


\texttt{∨ 3 ep: \textit{EmergencyPump} · ep.Switch = "On"]}

\textbf{Change propagation.} When a goal is deidealized, the change must be propagated along the refinement trees in which this goal is involved—both up and down such trees. The \texttt{AddConjunct} or \texttt{AddDisjunct} operators therefore have to be applied recursively to the other goals up and down refinement links.

For example, the integration of the countermeasure goal \texttt{Achieve[\textit{MineEvacuatedWhenPumpFailureAndPumpSwitchOn}]} requires change propagation to the descendants of the goal \texttt{Avoid[\textit{OverflowedMine}]. First, this goal is modified as previously shown. Next, the change is down-propagated, leading to an application of \texttt{AddConjunct} to the goal \texttt{Maintain[PumpOnWhenHighWater]}:

\texttt{Goal \textit{Maintain}[PumpOnWhenNoPumpFailureAndHighWater]}

\texttt{FormalSpec Vp: Pump, s: Sump s.WaterLevel = "High" \wedge PumpInSump(p, s) \wedge p.Failure = false \rightarrow p.Motor = "On"}

The next refinement instantiates the \textit{unmonitorability-driven} refinement pattern. The \texttt{AddConjunct} operator is therefore applied to the goal \texttt{Maintain[PumpOnWhenHighWaterDetected]}. We obtain:

\texttt{Goal \textit{Maintain}[PumpOnWhenNoPumpFailureAndHighWaterDetected]}


The next refinement instantiates the \textit{milestone-driven} refinement pattern. The \texttt{AddConjunct} operator is therefore applied to both children. We thereby obtain:

\texttt{Goal \textit{Maintain}[PumpSwitchOnWhenNoPumpFailureAndHighWaterDetected]}

\texttt{FormalSpec Vp: Pump, c: PumpController c.HighWaterSignal = "On" \wedge CtrlPump(c, p) \wedge p.Failure = false \rightarrow p.Switch = "On"}

\texttt{Goal \textit{Maintain}[PumpOnWhenNoPumpFailureAndPumpSwitchOn]}


Since these goals are leaf goals, the propagation ends.

Change propagation in the general case proceeds as follows. When \texttt{AddConjunct} or \texttt{AddDisjunct} is applied to a goal for deidealization, the pattern used for refining (resp. abstracting) it is identified. A \textit{propagation pattern} associated with the refinement/abstraction pattern tells us what goals in the refinement (resp. abstraction) must be modified and how. The process is applied recursively to the subgoals (resp. parents) until leaf goals (resp. root goals) are reached.

For example, if the \textit{case-driven} pattern is used for refining a goal through multiple disjoint cases [13], the application of \texttt{AddConjunct} or \texttt{AddDisjunct} to a child goal requires the application of the same operator to the parent goal (and vice-versa). If the \textit{milestone-driven} pattern is used for refining a goal through milestone subgoals [13], an application of \texttt{AddConjunct} to the first milestone subgoal requires the application of the same operator to the parent goal—but not necessarily to the other subgoals; the corresponding extra condition is not necessarily relevant to the latter.

For a given refinement pattern and application of \texttt{AddConjunct} or \texttt{AddDisjunct} to a goal, there might be alternative modifications of the refinement/abstraction structure. Moreover, other mechanisms are required for change propagation to goals not obtained through refinement patterns. Even though pattern-based propagation can be performed semi-automatically, the general problem of automatic change propagation through arbitrary refinement structures remains open.

VI. OBSTACLE RESOLUTION AS EXCEPTION HANDLING

The integration of countermeasure goals through new, explicit refinements in the original model raises several issues.

- The goal graph might undergo significant changes each time a new obstacle is identified.
- Normal situations would be mixed with exceptional ones; it might be hard to distinguish the former from the latter without domain expertise.
- Goal specifications become increasingly more complex.
- As new countermeasures are introduced, the ordered nesting of exceptional cases along refinements may lead to a combinatorial blow-up of special cases.

This section introduces a slight extension of the goal specification language that solves those issues. Dedicated constructs are provided for encapsulating the required modifications while documenting each exceptional case separately.

A. Extending the goal specification language

\textbf{Except.} A first construct links a countermeasure goal to its anchor goal:

\texttt{Goal AG}

\texttt{FormalSpec C \Rightarrow OT}

\texttt{Except \texttt{O then CG},}

where \texttt{AG} denotes the anchor goal \texttt{C \Rightarrow OT} of countermeasure \texttt{CG} to obstacle \texttt{O}. Semantically, this implicit specification is fully equivalent to the refinement in Fig. 4.

This construct may be used under the following precondition:

\texttt{(O, CG, Dom) \models AG}

For example, the goal \texttt{Avoid[MinersDrowning]} is satisfied in the ideal situation by avoiding mine overflow. Under the exceptional condition of a pump failure, the goal is guaranteed through miners evacuation. We may therefore write:

\texttt{Goal Avoid[MinersDrowning]}

\texttt{Except PumpFailure}

\texttt{then Achieve[MineEvacuatedWhenPumpFailureAndPumpSwitchOn]}

This specification is logically equivalent to the refinement illustrating the second integration schema in Section V.B. Multiple \texttt{Except} annotations may be attached to a single goal to cope with different obstacles; the latter may therefore be introduced incrementally. Compared with the complexity of an equivalent explicit specification, the complexity of an implicit goal specification with multiple \texttt{Except} annotations
remains linear in the number of exceptions. The specification of the ideal goal remains unchanged. Moreover, multiple annotations sharing the same countermeasure goal may be factored out to simplify the model.

**Provided.** A second construct specifies an extra conjunct on the antecedent of an ideal goal G to produce a countermeasure:

\[
\text{Goal G} \rightarrow \text{Provided EC,}
\]

where EC denotes an extra conjunct to be added to G's antecedent for resolving the considered obstacle. Semantically, this implicit specification is equivalent to:

\[
\text{Goal G} \rightarrow \text{ProvidedSpec } [C \land EC] \Rightarrow \emptyset T.
\]

The *Provided* construct is typically used for deidealizing goals. For example, the deidealization of the goal Maintain [PumpOn When PumpSwitchOn] may be specified by highlighting the normal situation as follows:

\[
\text{Goal} \text{ Maintain}[\text{PumpOnWhenPumpSwitchOn}]
\]

\[
\text{ProvidedSpec} \forall p: \text{Pump}
\]

\[
\text{p.Switch = "On" \land p.Motor = "On"}
\]

\[
\text{Provided} \ p.\text{Failure} \neq \false.
\]

It often proves convenient to write *ProvidedNot EC* instead of *Provided ~EC*.

**RelaxedTo.** Symmetrically to *Provided*, this construct specifies an extra disjunct on the consequent of an ideal goal G to produce a countermeasure:

\[
\text{Goal G} \rightarrow \text{RelaxedTo ED,}
\]

where ED denotes an extra disjunct to be added to G's consequent for resolving the considered obstacle. Semantically, this goal is equivalent to:

\[
\text{Goal G} \rightarrow \text{RelaxedSpec } [C \lor ED] \Rightarrow \emptyset T.
\]

This construct is useful for deidealizing goals as well. For example, another deidealization of the same goal Maintain [PumpOnWhenPumpSwitchOn] might be specified by highlighting the normal situation as follows:

\[
\text{Goal} \text{ Maintain}[\text{PumpOnWhenPumpSwitchOn}]
\]

\[
\text{RelaxedSpec} \forall p: \text{Pump}
\]

\[
\text{p.Switch = "On" \lor p.Motor = "On"}
\]

\[
\text{RelaxedTo ep: EmergencyPump \cdot ep.Switch = "On".}
\]

Multiple *Provided* and *RelaxedTo* annotations may be attached to a single goal to introduce multiple countermeasures.

**Replaces.** This construct appears useful for tracing previous versions of a goal:

\[
\text{Goal G} \rightarrow \text{Replaces G}
\]

Such traceability helps readers understand the rationale behind the final goal, e.g.,

\[
\text{Goal} \text{ Maintain}[\text{ThirdPartyPumpOnWhenPumpSwitchOn}]
\]

\[
\text{Replaces} \text{ Maintain}[\text{PumpOnWhenPumpSwitchOn}].
\]

### B. Exception diagrams

Textual goal specifications with *Except* and *Replaces* annotations may be graphically represented in an exception diagram. Fig. 5 shows a portion of such a diagram for the goal Maintain [ThirdPartyPumpOnWhenPumpSwitchOn]. This diagram captures that (a) when the obstacle *PumpFailure* occurs the countermeasure goal *Achieve [PumpRepairedAndOnWhen PumpFailureAndPumpSwitchOn]* will guarantee this goal, and (b) this goal replaces Maintain [PumpOnWhenPumpSwitchOn]. Note that the *Except* annotation has been propagated to the replacing goal.

VII. **MODEL REFACTORYING FOR ATTACHING OR DETACHING GOAL EXCEPTIONS**

In practice, the analyst should decide at some point whether a countermeasure goal refers to an exceptional situation or to a normal one to be considered in the ideal model. Such a decision might depend on various factors such as the frequency of the resolved obstacle, the criticality of the obstructed goal, domain-specific culture, stakeholders wishes, and so forth. To make the decision flexible and easily reversible, this section presents model refactoring operators for attaching or detaching the annotations introduced in Section VI to/from a goal model.

Three operators are available for transforming an annotated model portion into a standard one.

**Detach-Except**, applied to an annotated goal, produces a new model where the goal is no longer annotated with a specific *Except* clause. The operator introduces a new refinement with two children: the countermeasure goal and a deidealization of the original goal (see Fig. 4 from left to right). The children of the original goal are then children of the deidealized goal. Back to an earlier example, the operator takes the model fragment in Fig. 6a to produce the model fragment in Fig. 6b.

**Detach-Provided**, applied to an annotated goal, produces a new model where a specific *Provided* annotation is “compiled” into its equivalent formal specification. For example, after application of this operator the goal Maintain [PumpOnWhen PumpSwitchOn] is specified without its *Provided* annotation as follows:

\[
\text{Goal} \text{ Maintain}[\text{PumpOnWhenPumpSwitchOn}]
\]

\[
\text{ProvidedSpec} \forall p: \text{Pump}
\]

\[
[p.\text{Switch} = "On" \land p.\text{Failure} = \false] \Rightarrow p.\text{Motor} = "On"
\]

**Detach-RelaxedTo**, applied to an annotated goal, produces a new model where a specific *RelaxedTo* annotation is specified without its *RelaxedTo* annotation as follows:

\[
\text{Goal} \text{ Maintain}[\text{PumpOnWhenPumpSwitchOn}]
\]

\[
\text{RelaxedSpec} \forall p: \text{Pump}
\]

\[
[p.\text{Switch} = "On" \lor p.\text{Failure} = \false] \Rightarrow p.\text{Motor} = "On"
\]

\[
\text{RelaxedTo ep: EmergencyPump \cdot ep.Switch = "On".}
\]

**B. Exception diagrams**

Textual goal specifications with *Except* and *Replaces* annotations may be graphically represented in an exception diagram. Fig. 5 shows a portion of such a diagram for the goal Maintain [ThirdPartyPumpOnWhenPumpSwitchOn]. This diagram captures that (a) when the obstacle *PumpFailure* occurs the countermeasure goal *Achieve [PumpRepairedAndOnWhen PumpFailureAndPumpSwitchOn]* will guarantee this goal, and (b) this goal replaces Maintain [PumpOnWhenPumpSwitchOn]. Note that the *Except* annotation has been propagated to the replacing goal.

VII. **MODEL REFACTORYING FOR ATTACHING OR DETACHING GOAL EXCEPTIONS**

In practice, the analyst should decide at some point whether a countermeasure goal refers to an exceptional situation or to a normal one to be considered in the ideal model. Such a decision might depend on various factors such as the frequency of the resolved obstacle, the criticality of the obstructed goal, domain-specific culture, stakeholders wishes, and so forth. To make the decision flexible and easily reversible, this section presents model refactoring operators for attaching or detaching the annotations introduced in Section VI to/from a goal model.

Three operators are available for transforming an annotated model portion into a standard one.

**Detach-Except**, applied to an annotated goal, produces a new model where the goal is no longer annotated with a specific *Except* clause. The operator introduces a new refinement with two children: the countermeasure goal and a deidealization of the original goal (see Fig. 4 from left to right). The children of the original goal are then children of the deidealized goal. Back to an earlier example, the operator takes the model fragment in Fig. 6a to produce the model fragment in Fig. 6b.

**Detach-Provided**, applied to an annotated goal, produces a new model where a specific *Provided* annotation is "compiled" into its equivalent formal specification. For example, after application of this operator the goal Maintain [PumpOnWhen PumpSwitchOn] is specified without its *Provided* annotation as follows:

\[
\text{Goal} \text{ Maintain}[\text{PumpOnWhenPumpSwitchOn}]
\]

\[
\text{ProvidedSpec} \forall p: \text{Pump}
\]

\[
[p.\text{Switch} = "On" \land p.\text{Failure} = \false] \Rightarrow p.\text{Motor} = "On"
\]

**Detach-RelaxedTo**, applied to an annotated goal, produces a new model where a specific *RelaxedTo* annotation is specified without its *RelaxedTo* annotation as follows:

\[
\text{Goal} \text{ Maintain}[\text{PumpOnWhenPumpSwitchOn}]
\]

\[
\text{RelaxedSpec} \forall p: \text{Pump}
\]

\[
[p.\text{Switch} = "On" \lor p.\text{Failure} = \false] \Rightarrow p.\text{Motor} = "On"
\]

\[
\text{RelaxedTo ep: EmergencyPump \cdot ep.Switch = "On".}
\]
Similarly, three operators are available for transforming a standard model portion into an annotated one—namely, Attach-Except, Attach-Provided and Attach-RelaxedTo. These operators are the reverse of the Detach ones.

VIII. Evaluation

The techniques presented in this paper were applied\(^1\) to a benchmark commonly used for evaluating obstacle analysis techniques [1, 14, 15]. The goal and obstacle models used for the London Ambulance System (LAS) are based on [14].

The goal model contains 42 goals, 19 refinements instantiating a variety of refinement patterns, and 8 agents. The obstacle model contains 71 obstacles and 30 countermeasure goals. The full models can be found in [14, 15]. Only portions of the goal model are considered here.

The top goal in this model is Achieve [IncidentResolved]. The milestone-driven refinement pattern produces three subgoals:

- Achieve [IncidentReported], Achieve [AmbulanceOnScene\(\text{When IncidentReported}\)] and Achieve [IncidentResolved\(\text{When Ambulance OnScene}\)]. At a lower level of refinement, the goal Achieve [AmbulanceMobilized\(\text{When Allocated}\)] states that allocated ambulances shall be mobilized within 3 minutes. This goal is refined using a case-driven pattern into Achieve [Ambulance Mobilized At Station\(\text{When Allocated}\)] and Achieve [AmbulanceMobilized On Road\(\text{When Allocated}\)].

Countermeasures goals were used applying available resolution tactics [14]. For example, here are countermeasure goals for two leaf obstacles:

- Achieve [MobilizationOrderPrinted\(\text{And Ambulance Not Mobilized}\)]
- Achieve [MobilizationOrderTakenBy Other Ambulance]
- MDT-MobilizationOrderIgnored

The large number of obstacles and countermeasure goals called for our countermeasure integration and encapsulation techniques. As a result, the countermeasure goals appear to focus on a small number of important goals; e.g., the goal Achieve [IncidentResolved\(\text{By Ambulance Intervention}\)] has 15 exceptions. The overall integration produced 34 exceptions distributed over 7 goals only.

The techniques presented in this paper helped significantly for the following reasons.

Model simplification by separation of concerns. The goals referring to normal situations were systematically distinguished from those handling obstacle occurrences. Emerging assumptions were incrementally down-propagated to obstructed descendants of corresponding anchor goals; this required 7 propagations and produced 28 Provided annotations distributed over 6 goals. Without these annotations the formal specification of those 6 goals would have been cluttered with details related to exceptional cases. Table II quantifies our use of Provided annotations.

For example, the goal Achieve [AllocatedAmbulance \(\text{MobilizationWhen MobilizationOrder Printed}\)] is defined as follows after integration in the model:

- Goal Achieve [AllocatedAmbulance \(\text{MobilizedWhen Mobilization Order Printed}\)]
- Provided [AllocatedAmbulance Not Leaving Before Mobilization]
- Provided [AllocatedAmbulance Not Unavailable Before Mobilization]
- Provided [Printed Mobilization Order Not Ignored]
- Provided [Mobilization Not Taken By Other Ambulance].

The full equivalent specification of this goal without Provided annotations would completely hide the ideal case; it would then appear fairly hard to distinguish the part of the goal antecedent related to the ideal case from those related to exceptional cases.

The Detach-Except operator was applied to the case-driven refinement of the goal Achieve [AmbulanceMobilized \(\text{When Allocated}\)]. Allocating an ambulance when not at station was estimated fairly rare—5% of cases according to typical figures in the domain. The parent goal of these two goals was therefore modified accordingly:

- Goal Achieve [AmbulanceMobilized \(\text{When Allocated}\)]
- Except [AllocatedAmbulance Not At Station]

Such refactoring reduces model complexity by hiding the part of the model handling the mobilization of an ambulance when on road. The resulting ideal goal model therefore contains fewer refinements and fewer goals, making it easier to understand and clearly separate ideal behaviors from exceptional ones.

Compositionality. Without our techniques, the integration of so many exceptions for only 7 goals would have resulted in large, complex refinements with a combinatorial blow-up of special cases. To illustrate this important point, consider the goal Achieve [AmbulanceMobilized \(\text{When Allocated}\)]. Its original, ideal specification is:

\[
\forall \text{amb, inc}: \text{Incident Allocated(amb, inc)} \text{\Rightarrow } \text{\exists amb': \text{Ambulance } \cdot \text{Mobilized(amb', inc)}}
\]

After obstacle analysis, this goal is guaranteed through 5 countermeasure goals (see Fig 7). The brute-force integration of only the three countermeasure goals depicted at the bottom of Fig. 6 would have resulted in the following formal specification for the final version of the goal Achieve [AmbulanceMobilized \(\text{When Allocated}\)]:

\[
\forall c: \text{UrgentCall, inc: Incident Allocated(amb, inc)} \text{\Rightarrow } \text{\exists amb': \text{Ambulance } \cdot \text{Mobilized(amb', inc)}}
\]

In addition to this complex specification, the goal refinement structure would have been heavily modified:

- Achieve [AmbulanceMobilized \(\text{When Allocated}\)]
- Achieve [OtherAmbMobilized \(\text{When AllocatedAmbUnavailable}\)]
- Achieve [AllocAmbMobilized \(\text{When AmbAvailableUntilMob}\)]

\(^1\)See http://www.info.ucl.ac.uk/~acaillia/publications/las-system.html for full report.
With such a brute-force integration, each countermeasure goal must be refined by taking other countermeasures into account. This would lead to a combinatorial blow-up of cases. Thanks to our technique, the original specification of this goal and its refinement structure are preserved. The Except and Provided constructs encapsulate the modifications for a more robust system. Table I provides some figures on goal exceptions for other goals.

No premature decision and freedom of choice. The specification and documentation of exceptional behaviors was separated from the normal ones; this allowed us delaying the decision of how and when the handling of exceptional cases should occur.

Other benefits. The Replaces annotation was felt useful for documenting the replacing countermeasure goals – e.g., Achieve [Mobilized Ambulance Intervention Or Mobilization Cancelled] replacing Achieve [Mobilized Ambulance Intervention] to resolve the obstacle Mobilization Cancelled. Without this annotation we would have lost the previous version of the goal.

Exception diagrams significantly helped understand the model where all countermeasures are integrated; they document exceptions one single goal at a time (see Fig. 7). A total of 7 exception diagrams was produced for documenting exceptional cases and countermeasure goals.

Tool support. Our evaluation on the LAS case study was supported by a preliminary tool prototype. Given an obstacle resolution tactic and the corresponding anchor goal, the tool automatically generates the corresponding Except or Replaces annotations with corresponding countermeasure goal. The Provided and RelaxedTo constructs are supported as well. The tool also generates exception diagrams.

Fig. 7. Exception diagram for Achieve [Ambulance Mobilization]

IX. RELATED WORK

In the identify-assess-control cycles of risk analysis at requirements engineering time [8, 13, 14, 17, 20], most of the work so far has been devoted to risk identification and assessment. For risk identification, scenario-based heuristics are available [2, 29] as well as goal-oriented formal techniques [1, 14]. For risk assessment, various kinds of qualitative techniques are available [3, 5, 8, 25]. For risk control, the only work on countermeasure exploration is [14] where the obstacle resolution tactics mentioned in this paper are described. We are not aware of any work on systematic integration of countermeasures in a requirement model with a clear, precise semantics.

The relevance and importance of default-based reasoning has been recognized in the context of elaborating requirements or specifications. In [30], a formal framework is proposed for reasoning about evolving requirements. The framework is based on belief revision and default theory; operators for adding and retracting requirements are defined together with formal conditions for their valid application (similarly to our integration operators). The tracing of exceptional requirements is not discussed there. In [24], a specification is structured through axioms and Overides relations. Such relations are derived from the structural decomposition of the system. Specific axioms predominate more general ones when a conflict occurs. This framework comes with formal foundations and well-defined procedures for identifying conflicts and predominance among axioms. It appears more oriented towards specification elaboration. In [27], default specifications are introduced together with exceptions in order to increase the completeness of algebraic specifications; the But relation there somewhat corresponds to our Except relation.

Our approach mainly differs from those previous efforts in the following directions.

• Our techniques operate at requirements level and benefit from the refinement structure of a goal model. This structure helps in building a model where exception handling is integrated and in propagating required changes throughout the model.

• New requirements for a more robust system are incrementally integrated through obstacle analysis. The model updates are traceable back to the identified obstructed goals and their obstacles.

At programming level, aspects may be used for separating exception handling from normal code [19]. At modelling level, [28] convincingly shows how aspects can be used for separating exceptional behaviors from normal ones. As an alternative to the approach advocated in this paper, robustness

<table>
<thead>
<tr>
<th>Goals</th>
<th>Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achieve [Incident Resolved By Ambulance Intervention]</td>
<td>15</td>
</tr>
<tr>
<td>Achieve [Ambulance Mobilization]</td>
<td>6</td>
</tr>
<tr>
<td>Achieve [Allocated Ambulance Mobilization When Mobilization Order Printed]</td>
<td>5</td>
</tr>
<tr>
<td>Achieve [Mobilized Ambulance Intervention]</td>
<td>3</td>
</tr>
<tr>
<td>Achieve [Mobilized Ambulance Intervention Or Mobilization Cancelled]</td>
<td>3</td>
</tr>
<tr>
<td>Achieve [Allocated Ambulance Mobilization When Mobilization Order Displayed]</td>
<td>2</td>
</tr>
<tr>
<td>Achieve [Allocated Ambulance Mobilization At Station]</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goals</th>
<th>Provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achieve [Allocated Ambulance Mobilization On Road]</td>
<td>3</td>
</tr>
<tr>
<td>Achieve [Allocated Ambulance Mobilization At Station Based On Location Info]</td>
<td>4</td>
</tr>
<tr>
<td>Achieve [Allocated Ambulance Mobilization When Mobilization Order Printed]</td>
<td>4</td>
</tr>
<tr>
<td>Achieve [Allocated Ambulance Mobilization On Road Based On Location Info]</td>
<td>3</td>
</tr>
<tr>
<td>Achieve [Allocated Ambulance Mobilization When Mobilization Order Displayed]</td>
<td>3</td>
</tr>
<tr>
<td>Achieve [Allocated Ambulance Mobilization At Station]</td>
<td>1</td>
</tr>
</tbody>
</table>
aspects might be incorporated in a goal model by use of constructs similar to the ones sketched in, e.g., [10, 23]. Further work would however be required to define a declarative, logic-based semantics as well as an operational, trace-based semantics for such constructs—which seems unavailable to date. Suitable weaving mechanisms would then need to be defined in this semantic framework.

X. CONCLUSION

The paper presented systematic techniques for integrating countermeasures into ideal goal models. An integration operator was introduced as a model transformation ensuring progress towards a more complete model, minimal change of the original model, and refinement correctness preservation. Anchor goals were introduced to define where countermeasure goals should be integrated together with appropriate refinement schemas. Our goal-oriented RE framework was extended with constructs for structuring and documenting exceptional cases. Coming with these, model refactoring operators were proposed enabling analysts to attach and detach exceptions. The approach was evaluated on two case studies, a simple mine pump system and a much more complex ambulance dispatching system.

As shown in these case studies, a more complete goal model is obtained while the ideal model is kept visible. The ideal specifications are preserved. The final refinement structure turns out to be nearly the same as the original one. Exceptions are documented aside; analysts and users of the model can dive into independent exceptions one by one. A large number of countermeasure goals can be integrated; the integration techniques reduce model complexity by keeping the combinatorial blow-up of exceptional cases implicit.

The current version of our tool is fairly basic. Among the planned extensions, the increased automation of change propagation deserves highest priority. The propagation procedure itself should be made less dependent on common refinement patterns.

Complementary techniques are needed for selecting “best” countermeasures according to soft goals from the goal model. The responsibilities of agents in exception handling should be integrated at well. Moreover, the use of our exception-related constructs for deriving exception handlers in the corresponding software architecture would be worth investigating. In parallel, their exploitation for runtime self-adaptation in changing contexts appears a promising direction for future work.

ACKNOWLEDGEMENT

This work was supported by the EU Fund for Regional Development & the Walloon Region (TIC-FEDER Grant CE-IQS). We wish to thank B. Lambeau, C. Damas and S. Busard for discussions on our approach, and the reviewers for useful comments.

REFERENCES

