Reliability and Availability Requirements Engineering within the Unified Process using a Dependability Analysis and Modeling Profile

Simona Bernardi
Dipartimento di Informatica
Università di Torino
Torino, Italy
Email: bernardi@di.unito.it

José Merseguer
Dpto. Informática e Ing. Sistemas
Universidad de Zaragoza
Zaragoza, Spain
Email: jmerse@unizar.es

Robyn R. Lutz
Department of Computer Science
Iowa State University and JPL/Caltech
Ames, Iowa, USA
Email: rlutz@cs.iastate.edu

***Abstract*—In this paper, we propose an integration of the Unified Process and a Dependability Analysis and Modeling (DAM) profile to support quantifiable, testable reliability and availability requirements (R&AR) elicitation and completeness analysis. Specifically, we are interested in improved identification and specification of R&AR and their associated assumptions. This is accomplished through an iterative workflow that is consistent with the Unified Process and attaches DAM stereotypes to use and misuse cases. Fault-tree analysis helps determine and mitigate combinations of faults that could impede R&AR achievement. The workflow steps are demonstrated and evaluated on an intrusion protection service for critical infrastructures.**

***Index Terms*—Unified Process, reliability and availability requirements, Unified Modeling Language profile, misuse cases, fault trees.**

I. Introduction

Precise, accurate and feasible reliability and availability requirements that capture the customers' needs are difficult to construct without a structured process to understand the problem domain, detect and resolve conflicts among requirements, check for omissions and ambiguities and respond to change and new information [1], [2]. Iterative and incremental approaches such as the Unified Process handle functional requirements and the risks introduced by requirements evolution in software projects better than waterfall models [3]. However, the Unified Process itself pays little attention to non-functional requirements. The Unified Process uses the standard Unified Modeling Language [4] (UML) as its specification language, and UML profiles aim to extend its semantics and capabilities. Although there does not exist a UML standard profile tailored to the reliability and availability domains, some profiles (focused on software quality aspects) can help to gather non-functional requirements, most notably the Modeling and Analysis of Real Time and Embedded Systems (MARTE) profile [5]. The Dependability Analysis and Modeling profile [6] (DAM) is a specialization of MARTE mainly intended to support the analyst in verification and validation (V&V) activities for the dependability requirements of software systems. DAM considers a set of dependability *properties*, following the taxonomy in [7]: reliability, availability, maintainability, and safety. In this paper we focus on the first two of these, reliability and availability, although we anticipate that the approach proposed here would also assist with the requirements engineering of the other dependability properties.

The problem addressed by this paper is how to improve the elicitation, documentation and analysis of reliability and availability requirements (R&AR) within the context of the Unified Process. The approach that we propose is to extend the requirements workflow used to handle mainly functional requirements in the Unified Process to also handle R&AR. To accomplish this, we show how R&AR elicitation and documentation can be addressed within the Unified Process assisted by the DAM profile. The selection of the Unified Process provides a requirements engineering context and well-accepted elicitation techniques such as use cases to construct the necessary requirements artifacts. We show that the DAM profile offers an adequate syntax to both (1) specify R&AR in terms of quantifiable objectives or metrics and (2) to characterize faults and failures.

The primary contribution of the paper is an improved identification and specification of R&AR and their associated assumptions. This is accomplished through an iterative workflow that is consistent with the Unified Process and attaches DAM stereotypes to use and misuse cases. While the DAM profile has been used to support dependability verification and validation of software design, it has not previously been used to describe software requirements. We show that DAM can describe not only dependability mechanisms (e.g., redundant structures) but also threats to dependability, specifically in terms of hazards, failures, faults, and errors. Initially, much of this information involves assumptions about fault behavior based on domain knowledge and historical behavior of similar systems. As design progresses, additional information about the system fault behavior often emerges and the requirements may change accordingly.

One of the benefits of the approach described here is that it is incremental, so plans for the refinement over time of information about software and system dependability, as well as for changing requirements, can be made. In order to incrementally develop improved understanding and construct improved specification of the R&AR, fault tree analyses (FTAs) are incorporated into the requirements gathering process. The FTAs are used to gather information about the potential contributing causes of threats, to trace the combinations of faults and failures to use and misuse cases, and to explore mitigating strategies for removing these identified threats to reliability. Another benefit of our approach is that it overcomes a limitation of MARTE and DAM, i.e., the lack of guidelines on their usage in the context of software development processes. The paper provides a step-by-step process for developing software R&AR requirements specifications, presented in algorithmic form. The approach is further explained on an intrusion-tolerant, distributed firewall for critical information infrastructures developed within the CRUTIAL[1] project [8].

The paper is organized as follows. Section II presents the related work. Section III provides an overview of the Unified Process and the DAM profile, and introduces the case study. Section IV gives additional context for the requirements specification techniques used in this work. Section V presents our algorithmic requirements workflow for capturing R&AR. Section VI illustrates the approach through the case study. Finally, Section VII offers concluding remarks.

II. Related work

Mustafiz, et al. [9] extend use cases to model situations that can interrupt the system normal behavior. Use cases are mapped into a kind of probabilistic state-charts useful for quantitative assessment of safety and reliability. In [10], the authors express degraded services outcomes with annotations in use cases and others UML diagrams. They propose a requirement engineering process (DREP) to elicit these outcomes. Cortellessa and Pompei [11] propose UML extensions for the reliability analysis of component-based systems. In [12], a framework (UMD) has been proposed for eliciting and modeling dependability requirements that is designed around a basic modeling language defined by the authors. As in our proposal, UMD can be used to identify and make measurable the dependability requirements and properties of the system.

There exist several works [13], [14], [15], [16], [17] not specifically focused on R&AR but on safety elicitation, documentation or certification with UML. The work [13], although it does not associate hostile actors with use cases, as misuse cases do, describes potentially catastrophic failures. In particular, each use case is documented and failures are elicited in a table where each entry completely describes a hazard. The work in [14] inherits from [13] the use case specification and applies Practical Formal Specification to specify derived safety requirements as well as safety contracts. The work [15] proposes a UML profile to address safety requirements elicitation for aerospace software systems and the automatic generation of certification-related information. While the UML profile defined in [16] is meant for modeling safety-critical embedded real-time control systems. Finally, the methodology of Goseva, et al. [17] for risk assessment proposes to introduce the safety requirements in a tabular form, not in the UML diagrams as our proposal does.

Fault tree analysis is a widely researched technique with many variations [18]. Perhaps most relevant to the work described here is Software Fault Tree Analysis (SFTA). SFTA is a top-down, backward search technique that has been used successfully to find the possible software-related causes of a hazard or failure, e.g., on spacecraft [19]. SFTA has also been used to identify obstacles (such as false assumptions or unreliable behavior) to achieving the high-level goals (requirements) on an Unpiloted Aerial Vehicle [20]. The work [21] uses, in a synergistic manner, Fault Tree Analysis, Failure Mode and Effect Analysis (FMEA) and UML state-charts to derive, structure and integrate safety requirements of controlling software.

In the security domain, attack trees are a special case of fault trees, specialized to focus on security intrusions. Haley et al. [22] describe a general security requirements framework that also reasons backwards from the intrusion to its possible causes in an incremental manner. However, that work is primarily concerned with trust-related violations rather than with reliability requirements.

III. Basics and case study

First, we introduce the Unified Process and its relevant part to this work, the "requirements workflow". Next, we provide the basic knowledge of the DAM profile needed to understand our proposal. Finally, we present the case study.

A. Unified Process

The Unified Process [3] is a software development process that can be adapted to different kinds of software systems and application domains. This flexibility positions the Unified Process as an interesting choice to accommodate a R&AR stage for a broad number of software systems. The Unified Process uses UML [4], with its main features being: risk driven, architecture-centric, and iterative and incremental. Iterative means that the project breaks down into "iterations" that contain all the stages of a normal project, here known as "workflows" (requirements, analysis, design, implementation and test). Figure 1(a) shows the structure of the Unified Process, with an emphasis on the requirements workflow. Each workflow is visited in each iteration, which generates a partial version of the system. Iterations are grouped into "phases" (inception, elaboration, construction and transition) providing the macrostructure of the Unified Process. Workflows are orthogonal to phases; in each phase all workflows are tackled but with different emphases. For example, during the inception phase, more effort is devoted to the requirement workflow and less to the implementation.

The requirements workflow addressed by this work mainly occurs during the inception and elaboration phases; Figure 1(b) sketches its activities. The purpose is twofold: to discover what the system should do and to create a high-level specification for functional and non-functional requirements.

[1]IST project CRUTIAL (CRitical UTility InfrastructurAL resilience)

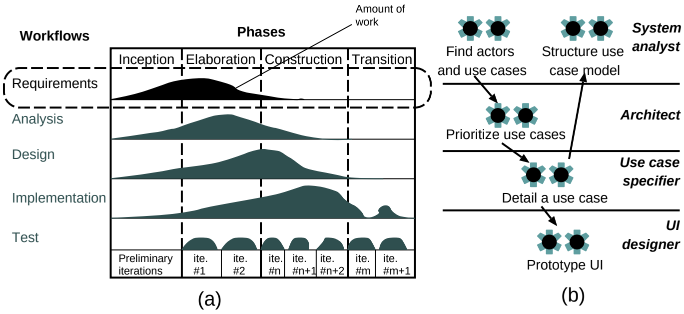

Fig. 1. Unified Process (taken from [3])

The Unified Process recognizes every project as unique, so different starting points for eliciting requirements may occur, from vague system ideas to initial versions already deployed. Therefore, the Unified Process advises different techniques and documents to cope with this heterogeneity, with it being up to the requirements engineer to select the best fit. The *feature list* is proposed to gather ideas that eventually may become real requirements. The *domain* and *business models*, expressed as UML class diagrams, describe and link those important concepts in the problem domain. *Use cases* are the first-class technique used to elicit requirements in the Unified Process. In Section IV-A we will describe some of their important aspects. The Unified Process advocates use cases also for specification of non-functional requirements (NFRs) within the description of the use case they affect. Moreover, the Supplementary Specification, described in Section IV-B, complements the non-functional specification. However, as in [23], we consider that use cases are not always the best choice to capture NFRs. Therefore, we propose for the first time the use of the DAM profile to complement the non-functional specification within the Unified Process.

B. DAM profile

The "Modeling and Analysis of Real Time and Embedded systems" (MARTE) profile [5] enables timing requirements specification in UML. MARTE is a *lightweight* extension (i.e., through the use of UML stereotypes and tagged-values) to support the modeling and analysis of systems that need to verify timing constraints. In particular, MARTE allows one to specify non-functional properties (NFPs) according to a well-defined Value Specification Language (VSL) syntax. MARTE was specialized for dependability modeling and analysis (DAM) in [6]. The DAM profile has been used so far to annotate dependability properties at the software design level, in particular, reliability, availability, maintainability and safety properties. DAM also has been applied to characterize processes leading to service failures and accidents by recording quantitative information such as probability of a fault's occurrence as well as qualitative information such as whether faults are permanent, transient, or intermittent, and the severity level assigned to each hazard (e.g., catastrophic, major, minor). The software design can ultimately be analyzed to compute metrics or to prove properties.

In this work we use DAM as a tool to help requirements engineers to capture dependability requirements, focusing on reliability and availability aspects. A DAM annotation *stereotypes* the model element it affects in the way UML proposes, i.e., by extending its semantics. In our approach, the extensions will specify a R&AR in the requirement artifacts of the Unified Process - i.e., the *domain model* and the *use case* diagram - and they will be attached to UML model elements (e.g., use cases in use case diagrams in Figures 6 and 8, and classes in the domain model in Figure 10). The entire set of DAM stereotypes (about fifteen), as well as the set of UML meta-classes the stereotypes can be applied to, can be found in [24]. A subset of them supports the specification of dependability properties (e.g., a *DaService* use case), while others can be used in the specification of fault-tolerance redundancy structures (e.g., a *DaVariant* class). Finally, some stereotypes can be used to characterize the threats affecting the modelled system (e.g., a *DaFaultGenerator* actor) and the recovery strategies (e.g., a *DaReplacementStep* action). Figure 2(a) depicts an excerpt of the DAM stereotypes used in this work.

According to UML, each DAM stereotype is made of a set of tags which define its properties. For example, *DaFaultGenerator* stereotype has *numberOfFaults* and *fault* as tags. The former is used to specify the number of concurrent faults, and the latter to characterize the nature of the fault. The types of the tags are basic UML types, MARTE NFP types (such as *NFP_integer*) or complex dependability types (such as *DaFault*). The latter are composed of attributes which may be MARTE NFP types or simple types, see Figure 2(b).

MARTE NFP types are data-types of special importance since they enable description of relevant aspects of the R&AR using properties such as: *value*, a value or parameter name; *expr*, a VSL expression; *source*, the origin of the NFP - e.g., a requirement (*req*), an assumed (*assm*), an estimated (*est*) or measured (*msr*) parameter; and *statQ*, the type of statistical measure (e.g., maximum, minimum, mean).

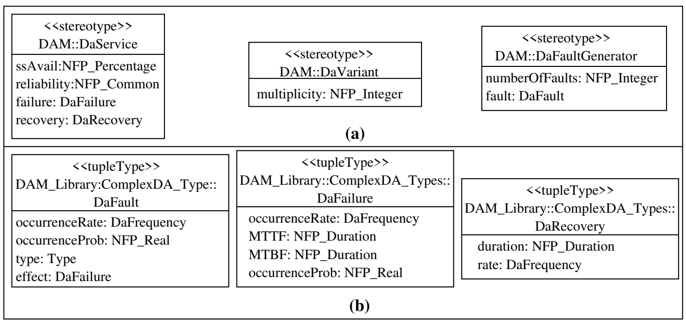

Fig. 2. DAM: (a) stereotypes (b) data types

C. An intrusion-tolerant firewall

The case study that we use to illustrate the proposed approach is the Crutial reference architecture to protect critical infrastructures [8], such as the Power grid, that models the infrastructure as a WAN-of-LANs. Figures 3(a) and (b), respectively, depict such an architecture and an initial domain model. The infrastructure, inside a trusted *LAN*, is made of *Hosts* that need to be protected. Only correct *messages* (according to a security policy) from the non-trusted *WAN* may reach the LAN hosts. A set of firewalls called Crutial Information Switches (*CIS*) aim to protect the LAN, even in the presence of accidents (e.g., node crashes) and attacks, thus providing perpetual and unattended correct system operation. A *traffic replicator* (e.g., an Ethernet hub) broadcasts the messages to all CIS Firewalls. These verify whether the message complies with the security policy and vote. An approved message is forwarded to its destination by the leader CIS Firewall. We

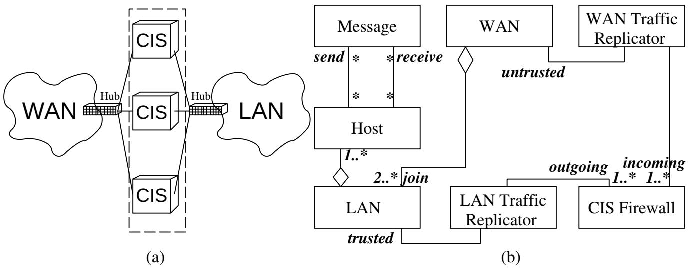

Fig. 3. (a) Architecture, (b) Domain Model

anticipate that this case study will be characterized by several NFRs, including reliability, availability and security, and may be affected by different classes of faults (e.g., physical faults like node crashes, malicious faults like DoS attacks). In this paper we focus on R&AR, and do not deal specifically with security aspects. Security aspects are not currently supported by DAM, but may be in the future.

IV. R&AR SPECIFICATION TECHNIQUES

We recall below the two techniques that our proposal exploits to gather and specify R&AR: use cases and the IEEE 830 standard. We selected them considering their appropriateness to be used along with the Unified Process. The case study, previously introduced, will be used as a running example.

A. Use Cases and Misuse Cases

Use cases [25] are a very well-known technique for eliciting requirements and likely the most popular. Indeed, the Unified Process is defined as a *use case driven* development process, which means that most of the workflow activities (requirements, analysis, design and implementation to test) are carried out starting from use cases. A use case diagram represents the different ways of using a system. It is made of actors and use cases, that are defined [4] as "a specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem or class can perform by interacting with outside actors". It is recognized that use cases, although an appropriate means to capture functional requirements, are not likewise suited for eliciting non-functional ones. To address this lack, *misuse cases* [26] were proposed for eliciting security, safety, reliability or availability requirements.

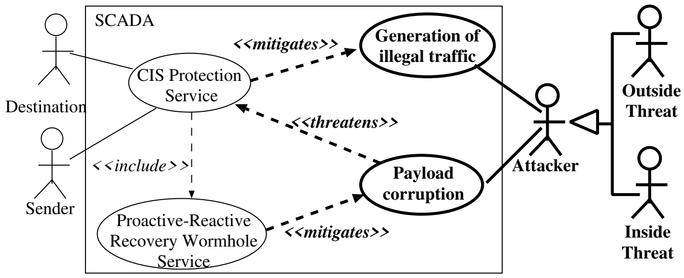

Fig. 4. Use Case Diagram

A misuse case is a use case from the point of view of a hostile actor. So in a use case diagram, as in Figure 4, use cases and actors can coexist with misuse cases and hostile actors (these latter are shown in bold in the figure). Once the analyst identifies a misuse case *threatening* the system, such as "Generation of illegal traffic", then s/he has to act to *mitigate* its effects. Usually, this takes the form of a new use case, e.g., "CIS Protection Service". Again, the hostile actors may find the way to threaten the system, e.g., with a "Payload corruption". So, *threat* and *mitigation* form a balanced zig-zag pattern of use and misuse cases that, respectively, define the ways to use and misuse the system. Use and misuse cases have to be detailed to accomplish the desired requirement specification. Although there exist several ways to do this, the Cockburn [27] template is a widely accepted one. Table I depicts the CIS Protection Service use case specification according to this template.

B. IEEE 830

The IEEE 830 standard [28] recommends approaches for software requirements specification (SRS) and describes the contents and qualities of a good SRS. However, this standard does not identify any specific method, nomenclature, or tool for preparing an SRS. Instead, what it discusses are the essential parts and information an SRS should address, here summarized in Figure 5(a). The third section, "Specific requirements", addresses the functional and non-functional requirements gathering, and different useful templates are suggested for its realization. Among them, we selected that in Figure 5(b), which has actually inspired the Supplementary Specifications suggested by the Unified Process [29]. The Unified Process proposes to describe those NFRs related to a specific use case as a "Special Requirements" section in the use case description. Those system requirements that are not readily specified in the use cases, should instead be specified in the Supplementary Specifications. Indeed, they are mainly those non-functional requirements (NFRs) that relate either to the system level or to several use cases. The Unified Process thus uses the Supplementary Specifications and the "Special Requirements" sections as complementary means to document NFRs.

Table of Contents

1. Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms, and abbreviations
   1.4 References
   1.5 Overview
2. Overall description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 Constraints
   2.5 Assumptions and dependencies
3. Specific requirements
Appendixes
Index

(a)

Template of "Specific requirements"

3. Specific requirements
   3.1 External interface requirements
      3.1.1 User interfaces
      3.1.2 Hardware interfaces
      3.1.3 Software interfaces
      3.1.4 Communications interfaces
   3.2 Functional requirements
   3.3 Performance requirements
   3.4 Design constraints
   3.5 Software system attributes
      3.5.1 Reliability requirements
      3.5.2 Availability requirements
      3.5.3 Security requirements
      3.5.4 Maintainability requirements
      3.5.5 Portability requirements
   3.6 Other requirements

(b)

Fig. 5. Software requirement specification outline (taken from [28])

V. A PROPOSAL FOR ELICITING R&AR

This section presents our proposal to gather and specify R&AR, which is placed within the requirements workflow of the Unified Process and should be applied during the inception and elaboration phases (Figure 1(a)). Among the requirement workflow artifacts, our proposal manages the domain model, the supplementary specifications and the use case diagram, which includes use and misuse cases. The DAM profile is used together with them to specify the R&AR.

The steps needed to carry out the R&AR elicitation in a generic iteration i of the requirement workflow are shown in Algorithm 1. The algorithm proposes a workflow that does not constrain other activities in the requirement workflow (Figure 1(b)), but complements them. The algorithm applied in the i^{th} iteration of the requirement workflow requires artifacts from a previous iteration $i-1$.

In the following, we discuss the relevant aspects of the algorithm. At the beginning (line 1), new use cases, misuse cases and actors can be discovered, e.g., by considering the current domain model. There are several systematic ways to discover misuse cases, although no technique can assure completeness. Successful techniques include guideword-based inquiries for each use case [13], scenario-based searches [26], checklist-based investigations of historically troublesome features [30], automated contingency analysis [20], and use of completeness criteria [31]. Depending on the domain, multiple techniques have been used in combination. The process of identifying misuse cases is incremental and ongoing to reflect both increased understanding over time and the evolution of the system and its environment.

In the next step (line 2), the system analyst selects a subset of use cases of interest. It may happen that some of them were already elaborated in previous iterations. Use case selection criteria should consider, among others, the risk level associated to UC realization.

The *Specify* activity (line 4) consists in describing the target use case and annotating it with the DAM profile. The use case description is accomplished by using the Cockburn

---

Algorithm 1 i^{th} iteration in the requirements workflow.

Require: Domain Model (DM_{i-1}), Use Case Diagram (UCD_{i-1}), Supplementary Specification (SS_{i-1})
Ensure: DM_i, UCD_i, SS_i
1: Discover new use cases, misuse cases and actors: $UCD_i \leftarrow UCD_{i-1} \bigcup UC_i^{new} \bigcup MUC_i^{new} \bigcup Act_i^{new}$
2: Select use cases to be specified: $selUC_i \subseteq UCD_i$
3: **for all** $uc \in selUC_i$ **do**
4:    Specify(uc) {*DAM annotations*}
5: **end for**
6: Select misuse cases related to $selUC_i$: $selMUC_i \subseteq UCD_i$
7: **for all** $muc \in selMUC_i$ **do**
8:    Specify(muc) {*DAM annotations + Fault trees*}
9: **end for**
10: Discover new NFRs: $SS_i \leftarrow SS_{i-1} \bigcup NFR^{new}$
11: Select a subset of requirements: $selNFR_i \subseteq SS_i$
12: **for all** $nfr \in selNFR_i$ **do**
13:    Elaborate(nfr) {*DAM annotations*}
14: **end for**
15: Restructure UCD_i and DM_i if necessary

---

template [27] (e.g., Table I describes the CIS Protection Service use case), which is characterized by several sections. Among them, the "Special Requirements", already introduced in Section IV-B, includes the NFRs directly related to the use case. When R&AR are related to the use case, the DAM profile is applied to rewrite them in a standard and disciplined form and to attach them as notes to the use case. For example, the light grey annotation in Figure 6 rewrites the R&AR in Table I. The use case, modeling a high-level service, is then stereotyped as *DaService* and proper VSL expressions may set to one or more tags (Fig. 2(a)), for example:

- The steady state availability (**ssAvail**), that is the percentage of service uptime;
- The probability of service delivery during a time period (**reliability**), that is a survival function;
- The Mean Time To Failure (**failure.MTTF**) and the Mean Time Between Failures (**failure.MTBF**);
- The duration of service recovery (**recovery.duration**).

The next step (line 6) selects misuse cases related to the former use cases via *threatens* or *mitigates* relationships. As previously, the *Specify* activity (line 8) describes the misuse case with the Cockburn template and annotates it with the DAM profile. However, in this case, the section "Success guarantee" indicates the possible failure modes; the "Main/Alternate scenarios" sections are used to describe failure scenarios; and the " Special Requirement" section may give further fault/failure characterization. Finally, the "Relationships" section describes relationships between the misuse case and use cases. Table II shows the description of the misuse case "Payload corruption".

In addition to this informal description of misuse cases, fault trees can also be used to investigate and more formally specify actions of the negative actors that may lead the misuse case to success. The root of the fault tree represents the description in the "Success guarantee" section and its leaf nodes model trigger actions from hostile actors. Fault trees can also formalize relationships between a misuse case and related use cases, as exemplified in Figure 7.

Since misuse cases and hostile actors represent sources of faults for some system services then, using DAM, they can be stereotyped as *DaFaultGenerator* (Figure 2(a)). Such stereotyping enables the analyst to characterize faults and their effects on system functionalities (e.g., failures) through the **numberOfFaults** (i.e., the number of concurrent faults) and **fault** tags. The latter is a complex data-type (Figure 2(b)) made of several tags, such as: **type** (fault classification according to [7]), **occurrenceRate** and **occurrenceProb** (fault occurrence rate and probability), and **effect** (failures caused by the fault).

Finally, those NFRs not related to a single use case will be reported in the supplementary specifications (line 10). A subset of them (line 11) will be either elaborated or refined (line 13) in this iteration. In particular, when the NFR addresses redundancy aspects, then the DAM profile can be used to identify the redundant structures in the domain model (e.g., Fig. 10).

VI. Application to the case study

In this section we apply Algorithm 1 to the case study for two initial iterations in the elaboration phase. The purpose is to illustrate how to incrementally identify R&AR as well as how to specify them with the DAM profile.

A. Elaboration phase - Iteration 1

We assume that during the inception phase some requirements were collected in a feature list and that a domain model was created (Fig. 3(b)). The feature list was elaborated to represent the functional requirements in a use case diagram (Fig. 4), while the NFRs not related to a specific use case were reported in the supplementary specifications. In the current iteration, no new elements are discovered (Algorithm, line 1). The CIS Protection Service use case, that represents the SCADA (Supervisory Control And Data Acquisition) protection service for hosts in a LAN, is selected for specification (line 2) since it provides basic system functionality. Its specification (line 4) conveys a first use case detailed description shown in Table I. The use case is stereotyped as a *DaService* (see Fig. 6) and, from the "Special Reqs" section, a DAM annotation is completed with two tagged-values: **ssAvail**, that traces to *A1*, and **failure.MTBF**, that traces to *R1*. Their values represent a minimum threshold.

The "Payload corruption" and "Generation of illegal traffic" misuse cases are the ones related to the CIS Protection Service use case. The first threatens it while the latter is mitigated by the use case. Both are selected (line 6) to be specified (line 8) in this iteration. Table II sketches the "Payload corruption" specification; for space reasons only the relevant parts for

UC Name	CIS Protection Service
Scope	SCADA
Main Actors	Sender (computer from the WAN), Receiver (computer of the protected LAN)
Success guarantee	The correct message is eventually delivered. The illegal message is not delivered.
Main scenario	A message is sent by Sender to Receiver: 1. It arrives at the CIS Firewalls 2. Each CIS Firewall checks if it satisfies the security policy and votes 3. The CIS Firewalls agree upon a final judgment (*majority voting*) 4. The message is correct and the CIS Firewall leader forwards it to the Receiver.
Alternate scenarios	4a. The message is illegal, then it is not delivered.
Special Reqs	*A1*. The CIS Protection Service shall be available 99.99% of the time. *R1*. The MTBF (Mean Time Between Failures) shall be at least 6 months.
Relationships	- CIS Protection Service *includes* Proactive-Reactive Recovery Wormhole Service - Payload corruption *threatens* CIS Protection Service - CIS Protection Service *mitigates* Generation of illegal traffic

TABLE I
CIS Protection Service use case

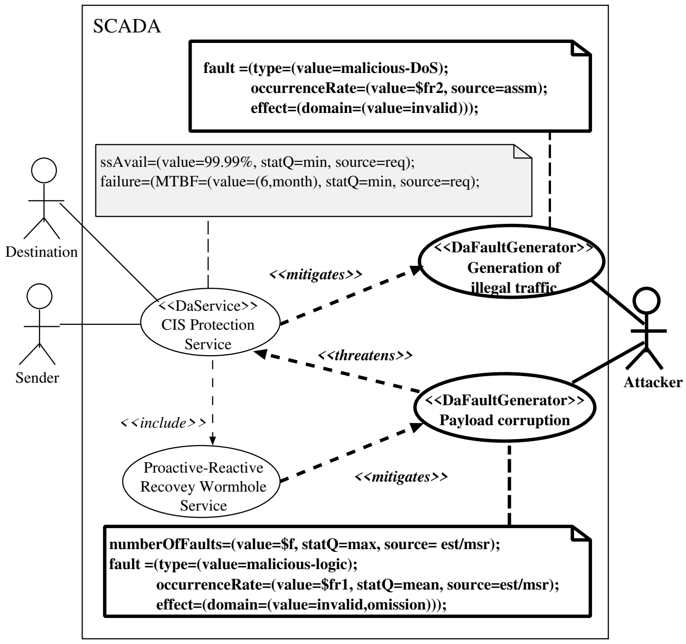

Fig. 6. Use case diagram - end of iteration 1.

getting DAM annotations are shown. The latter are shown in bold in Fig. 6.

We have used a fault tree to specify how "Payload corruption" threatens the CIS Protection Service (Fig. 7). The CIS Protection Service fails when the majority of the payloads ($\lceil n/2 \rceil + 1$) becomes corrupted, at which point either the quorum cannot be reached or a wrong judgment is made, or the payload leader fails to forward the approved message to the destination. Observe that, although the leaves of the fault tree are left unexplored, they could be refined considering “Main/Alternate scenarios” in this misuse case, following the approach in [32].

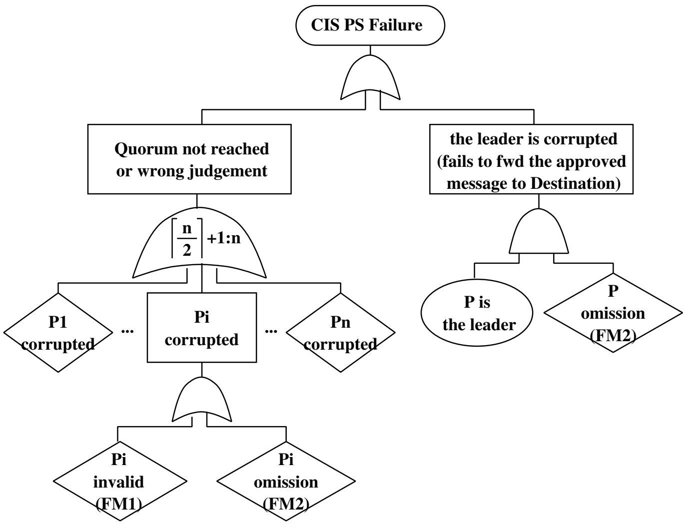

Fig. 7. The Payload corruption threatens the CIS Protection Service specified by a fault tree

As explained in Sect. V, the misuse cases are stereotyped as *DaFaultGenerator* (Fig. 6) and the “Success guarantee”, “Main scenario” and “Special Reqs” sections help to define the tagged-values of the stereotyped misuse cases.

In particular, several tagged-values are annotated for the misuse case “Payload corruption”. The **fault.effect** traces the payload failure modes of the “Success guarantee”, with respect to the failure domain: invalid (*FM1*) and omission (*FM2*). The **fault.type** comes from the “Main scenario” and characterizes, from a qualitative point of view, the type of fault affecting the payload. Finally, **numberOfFaults** and **fault.OccurrenceRate** are two parameters ($\$f$ and $\$fr1$) to be estimated/measured during the validation and verification activities or the test activities. They trace, respectively, to *F1* and *F2* in “Special Reqs”. The former specifies the maximum number of concurrent faults to be considered possible, while the latter specifies the mean rate of fault occurrence for a payload. Similar DAM annotations are attached to the misuse case “Generation of illegal traffic” (Figure 6). In this case, the **fault.type** is a Denial of Service (DoS) attack and the **fault.OccurrenceRate** tagged-value is an assumed parameter ($\$fr2$) to be set during the validation and verification activities according to the network bandwidth characteristics.

In the next step of the algorithm (line 10), a new NFR is discovered and reported in the supplementary specification (Table III). The NFR is actually a fault tolerance requirement related to the system redundancy level and it is selected to be elaborated in the current iteration (lines 11,13). The fault tolerance requirement stems from 1) the assumption on the maximum number of concurrent faults (Table II, “Special Reqs” *F1*) and 2) the majority voting policy specified, as a requirement, in the CIS Protection Service use case (Table I, Main Scenario, step 3).

3.6 Other requirements
(*Fault Tolerance*) There shall be at least $2f+1$ CIS Firewalls to tolerate f concurrent faults.

TABLE III
SUPPLEMENTARY SPECIFICATION EXCERPT

Such an NFR suggests stereotyping the CIS Firewall class in the domain model (Fig.3) as *DaVariant*, where each CIS Firewall object represents a component with a different design that provides the same service. The NFR is then formalized by the expression associated to the **multiplicity** tag as follows:

multiplicity = (value=$\$n$, expr=($\$n \geq 2 * \$f + 1$), source=req)

where $\$n$ is a parameter that represents the redundancy level. The use case diagram (Fig. 6) and the domain model (not shown), although incremented in this iteration, do not need to be restructured, so the last step in Algorithm 1 is not performed. Both, use case diagram and the domain model, are inputs for the other workflows of the Unified Process (e.g., design, test) as well as for the validation and verification activities.

Observe that, as for test activities, validation and verification activities can also be planned by tracing to the considered use cases and misuse cases, to be eventually realized once the CIS Protection Service design becomes available. For example, an activity could trace to the misuse case “Payload corruption” with the objective of providing an estimation of the mean fault occurrence rate $\$fr1$.

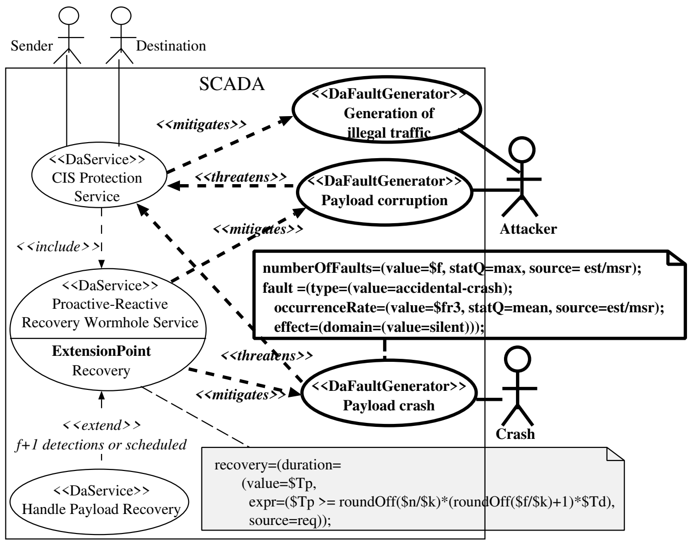

Fig. 8. Use case diagram - end of iteration 2.

B. Elaboration phase - Iteration 2

In the current iteration, a new hostile actor “Crash”, a new misuse case “Payload crash” and two new use cases

MUC Name	Payload corruption
Scope	CIS Protection Service
Main Actors	Attacker: Ouside Threat, Inside Threat
Success guarantee	The Payload evaluates as "correct" an illegal message or it evaluates as "illegal" a correct message (*FM1*), or it is subjected to temporary omission (*FM2*).
Main scenario (Outside Threat)	The Attacker identifies the WAN traffic replicator as potential target: 1. The Attacker sniffs the network traffic 2. The Attacker gets unauthorized access to a host in the LAN 3. The Attacker installs *malicious logic* in the accessed host 4. The hosted Payload behaves in an unpredicted manner.
Special Reqs	*F1*. At most f Payloads can be concurrently corrupted. *F2*. f should be set according to the mean rate of fault occurrence.
Relationships	Payload corruption *threatens* CIS Protection Service (fault tree Fig.7)

TABLE II
PAYLOAD CORRUPTION SPECIFICATION

"Proactive-Reactive Recovery Wormhole Service" and "Handle Payload Recovery" have been found. All are added to the current use case diagram (Figure 8). We have then selected for specification the CIS Protection Service and the Proactive-Reactive Recovery Wormhole Service (lines 2, 4).

The CIS Protection Service description is updated, in the "Relationships" section, with the new threatening misuse cases. The use case Proactive-Reactive Recovery Wormhole Service introduces an improved service to mitigate the "Payload corruption" and it is specified in Table IV.

The "Special Reqs" section includes a lower bound for the recovery period T_p as a requirement (*IT1*). This NFR is related to the CIS Protection Service availability since it defines the minimum timing window an attacker has to compromise more than f payloads. Moreover, T_p is a function of the worst case execution time T_d of a payload recovery. The use case is then stereotyped as *DaService* and annotated with the **recovery.duration** tagged value, which traces to *IT1* (Figure 8).

The misuse cases "Payload corruption" and "Payload crash" are selected (lines 6,8) in the current iteration. The former is updated with information related to mitigation activities performed by the Proactive-Reactive Recovery Service use case. The specification of the latter (not shown here for space reasons) includes a fault tree that formalizes its *threatens* relationship with the CIS Protection Service. This fault tree and the one associated with "Payload corruption" (Figure 7) have then been integrated to model their combined effects on the CIS Protection Service (Figure 9). Moreover, from the "Payload crash" specification a new DAM annotation raises. This is similar to the one associated to the "Payload corruption" (compare Figures 6 and 8). In particular, observe that the same parameter $\$f$ is used to specify the maximum number of payload faults (either crash or corruption).

In the next step (line 10) no new R&AR are discovered. However, the fault tolerance requirement specified in Table III has been elaborated (line 13) to take into account the new features of the Proactive-Reactive Recovery Service included in the CIS Protection Service:

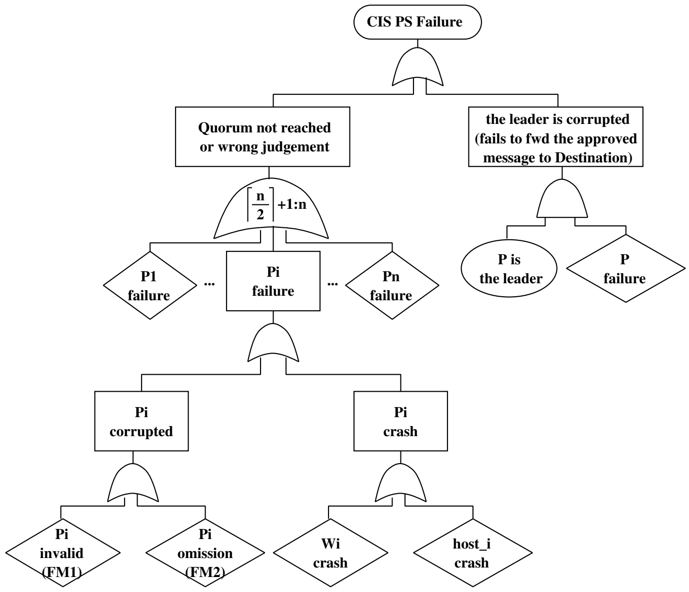

Fig. 9. Relationships between the CIS Protection Service use case and the threatening misuse cases.

a) 3.6 Other reqs (Fault Tolerance): There should be at least $2f + k + 1$ CIS Firewalls to tolerate f concurrent faults and k parallel Payload recoveries in a recovery period T_p.

This change leads to a restructuring of the domain model (Figure 10). In particular, the DAM annotation associated to the CIS Firewall class of the domain model has been refined. Finally, the use case diagram is restructured (line 15) to show explicitly the *extend* relationship between the use cases "Handle Payload Recovery" and "Proactive-Reactive Recovery Wormhole Service".

VII. CONCLUSION

In this work we have addressed the problems of eliciting and analyzing R&AR from a software reliability engineering perspective. We chose to use well-accepted practices and standards in the software engineering field to improve the solutions' generality. Thus, the Unified Process and uses cases have provided a methodological framework, while a

UC Name	Proactive-Reactive Recovery Wormhole Service
Scope	CIS Protection Service
Pre-conditions	a) Event-trigger (reactive recovery), b) Time-trigger (proactive/reactive recovery)
Success guarantee	Up to f concurrent faults/intrusions are tolerated
Main scenario	Payload P_i notifies to Wormhole W_j that P_j is either suspected or detected to be faulty (i.e., crash/corrupted): 1. If the number of collected suspicions/detections is at least $f+1$ then W_j executes the Recovery of its Payload P_j: 1a. Immediately, if there are $f+1$ detections 1b. In the next rejuvenation period, coordinated with the periodic proactive recovery, otherwise.
Alternative scenario	A new recovery period T_p starts: 1. All the Wormholes synchronize their clock 2. The Wormholes with their Payloads suspected/detected to be failed execute the scheduled Recovery of their Payloads. 3. Up to k different Wormholes execute the periodic Recovery of their Payloads.
Extension Points	Handle Payload Recovery use case realizes the recovery.
Special Reqs.	*IT1*. The recovery period shall be at least $\lceil n/k \rceil * (\lceil f/k \rceil + 1) * T_d$, where T_d is the worst case execution time of a Payload recovery.
Relationships	CIS Protection Service *includes* Proactive-Reactive Recovery Wormhole Service Proactive-Reactive Recovery Wormhole Service *mitigates* Payload corruption Proactive-Reactive Recovery Wormhole Service *mitigates* Payload crash

TABLE IV
PROACTIVE-REACTIVE RECOVERY WORMHOLE SERVICE USE CASE

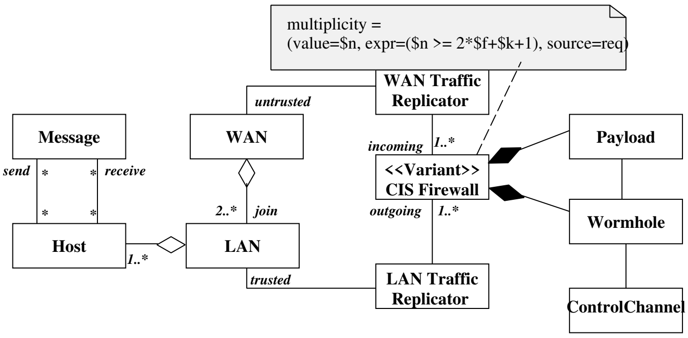

Fig. 10. Domain model - end of iteration 2.

dependability profile (DAM) has been used for an improved specification of R&AR. The combination of these with fault tree analysis helped to determine faults and events threatening R&AR achievement.

The use case diagram and the domain model, annotated with DAM and produced in a given iteration, provide input for the other workflows of the Unified Process (e.g., analysis & design, test), as well as for validation and verification activities. In particular, a set of activities, that trace to a (subset) of (mis)use cases of the use case diagram, can be planned in advance, similarly to *test cases* in the Unified Process test workflow. Such planned activities will be carried out as the design specification becomes available.

The approach presented in this paper can be extended in several directions. Future work includes the study of the DAM profile applicability in the other workflows of the Unified Process. Moreover, regarding validation and verification activities, we plan to define model transformations that take as input UML+DAM models, created in the requirement and design stages, and produce different models for the R&AR assessment.

ACKNOWLEDGMENT

Simona Bernardi thanks project PACO (Performability-Aware Computing: Logics, Models, and Languages) funded by the Italian Ministry of University and Research.

José Merseguer thanks project DPI2006-15390 of the Spanish Ministry of Science and Technology.

Robyn Lutz thanks the U.S. National Science Foundation, grants 0541163 and 0916275, for partial support of this work.

REFERENCES

[1] M. Donnelly, B. Everett, J. Musa, and G. Wilson, "Best current practice of software reliability engineering," in *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. IEEE Computer Society Press, 1996, ch. 6, pp. 219–254.

[2] A. Abran, J. W. Moore, P. Bourque, and R. Dupuis, Eds., *Guide to the Software Engineering Body of Knowledge*. Los Alamitos, California: IEEE Computer Society, 2004. [Online]. Available: http://www.swebok.org/

[3] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Addison Wesley, 1999.

[4] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, 2nd ed. Addison Wesley, 2004.

[5] *A UML profile for Modeling and Analysis of Real Time Embedded Systems, Beta 1*, Object Management Group, August 2007, adopted Spec., ptc/07-08-04.

[6] S. Bernardi, J. Merseguer, and D. Petriu, "A dependability profile within MARTE," *Software and Systems Modeling*, August 2009, DOI: 10.1007/s10270-009-0128-1.

[7] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 01, no. 1, pp. 11–33, 2004.

[8] P. Verissimo, N. Neves, M. Correia, A. A. E. Kalam, and A. Bondavalli, "The CRUTIAL architecture for critical information infrastructures," in *Architecting Dependable Systems V*, ser. LNCS 5135, R. de Lemos et al., Ed., Berlin Heidelberg, 2008, pp. 1–27.

[9] S. Mustafiz, X. Sun, J. Kienzle, and H. Vangheluwe, "Model-driven assessment of system dependability," *Journal of Software and Systems Modeling*, vol. 7, no. 4, pp. 487–502, 2008.

[10] S. Mustafiz, J. Kienzle, and A. Berlizev, "Addressing degraded service outcomes and exceptional modes of operation in behavioural models," in *Proc. of the RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems (SERENE'08)*. New York, NY, USA: ACM, 2008, pp. 19–28.

[11] V. Cortellessa and A. Pompei, "Towards a UML Profile for QoS: a contribution in the reliability domain," in *Proceedings of the Fourth International Workshop on Software and Performance (WOSP'04)*, January 2004, pp. 197–206.

[12] P. Donzelli and V. Basili, "A practical framework for eliciting and modeling system dependability requirements: Experience from the NASA high dependability computing project," *Journal of System and Software*, vol. 79, pp. 107–119, 2006.

[13] K. Allenby and K. Kelly, "Deriving safety requirements using scenarios," in *International Conference on Requirements Engineering*. IEEE Computer Society, 2001, pp. 228–235.

[14] F. Iwu, A. Galloway, J. McDermid, and J. Toyn, "Integrating safety and formal analyses using UML and PFS," *Reliability Engineering and System Safety*, vol. 92, no. 2, pp. 156–170, 2007.

[15] G. Zoughbi, L. Briand, and Y. Labiche, "A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B), Safety-Critical Software," in *Proceedings of Models 2007*, ser. LNCS. Springer-Verlag, 2007, vol. 4735, pp. 574–588.

[16] L. Shourong and A. Wolfgang, "A UML Profile to model safety-critical embedded real-time control systems," in *Studies in Computational Intelligence (SCI)*. Springer-Verlag Berlin Heidelberg, 2007, vol. 42, pp. 197–218.

[17] K. Goseva-Popstojanova *et al.*, "Architectural-level risk analysis using UML," *IEEE Transactions on Software Engineering*, vol. 29, no. 10, pp. 946–960, 2003.

[18] M. R. Lyu, Ed., *Handbook of Software Reliability Engineering*. IEEE Computer Society Press, 1996.

[19] R. Lutz and A. Nikora, "Failure Assessment," 1st Int'l Forum on Integrated System Health Engineering and Management for Aerospace (ISHEM 2005), November 2005.

[20] R. Lutz, S. Nelson, A. Patterson-Hine, C. Frost, D. Tal, and R. Harris, "Using obstacle analysis to identify contingency requirements on an unpiloted aerial vehicle," *Requirements Engineering*, vol. 12, no. 1, pp. 41–54, 2007.

[21] E. Troubitsyna, "Elicitation and specification of safety requiments," in *IEEE Proc. of the Third Int'l Conference on Systems*. IEEE, 2008, pp. 202–207.

[22] C. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh, "Security requirements engineering: A framework for representation and analysis," *IEEE Transactions on Software Engineering*, vol. 34, no. 1, pp. 133–153, 2008.

[23] J. Arlow and I. Neustadt, *UML 2 and the Unified Process*, 2nd ed. Addison Wesley, 2005.

[24] S. Bernardi, J. Merseguer, and D. Petriu, "A UML profile for Dependability Analysis and Modeling of Software Systems," Universidad de Zaragoza, Spain, Tech. Rep. RR-08-05, 2008.

[25] I. Jacobson, M. Christenson, P. Jonsson, and G. Overgaard, *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, 1992.

[26] I. Alexander, "Misuse cases: Use cases with hostile intent," *IEEE Software*, vol. 20, no. 1, pp. 58–66, 2003.

[27] A. Cockburn, *Writing Effective Use Cases*. Addison Wesley, 2001.

[28] IEEE Std 830-1998, "IEEE recommended practice for software requirements specifications," 1998.

[29] "Unified Process for EDUcation," http://www.upedu.org/upedu/, 2008.

[30] R. R. Lutz, "Targeting safety-related errors during software requirements analysis," in *SIGSOFT FSE*, 1993, pp. 99–106.

[31] N. Leveson, *Safeware*. Addison-Wesley, 1995.

[32] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, and R. Lutz, "A Software Fault Tree approach to requirement analysis of an intrusion detection system," *Requirement Engineering*, vol. 7, pp. 207–220, 2002.