PostPrint version
():1–13

A model-driven approach to survivability requirements assessment for critical systems

Simona Bernardi[1]*, **Lacramioara Dranca**[1] **and José Merseguer**[2]
[1]*Centro Universitario de la Defensa, Academia General Militar, Spain,*
[2]*Dpto. Informática e Ing. de Sistemas, Universidad de Zaragoza, Spain*

Abstract

Survivability is a crucial property for those systems -such as critical infrastructures or military Command and Control Information Systems (C2IS)- that provide essential services, since the latter must be operational even when the system is compromised due to attack or faults.

This paper proposes a model-driven method and a tool -*MASDES*- to assess the survivability requirements of critical systems. The method exploits the use of: 1) (mis)use case technique and UML profiling for the specification of the survivability requirements, and 2) Petri Nets and model-checking techniques for the requirement assessment. A Survivability Assessment Model (SAM) is obtained from an improved specification of misuse cases, which encompasses essential services, threats and survivability strategies. The SAM is then converted into a Petri Net model for verifying survivability properties through model checking.

The MASDES tool has been developed within the Eclipse workbench and relies on Papyrus tool for UML. It consists of a set of plug-ins that enable: 1) to create a survivability system view using UML and profiling techniques, and 2) to verify survivability properties. In particular, the tool performs model transformations in two steps. First, a model-to-model transformation generates, from the survivability view, a Petri net model and properties to be checked in a tool-independent format. Second, model-to-text transformations produce the Petri net specifications for the model-checkers. A military C2IS has been used as a case study to apply the method and to evaluate the MASDES tool, within an iterative-incremental software development process.

Keywords
Misuse cases, UML profiling, Petri Net, model-checking, survivability, model-to-model and model-to-text transformations

1. Introduction

Critical systems offer services that must operate safe and timely, despite the presence of faults or attacks. This is the case, for example, of critical infrastructures or military Command and Control Information Systems (C2IS). The latter share information and send orders to synchronize *essential assets* timely. Essential assets, supported by *essential services*, are

* Corresponding author; e-mail: simonab@unizar.es

defined as those critical system capabilities to fulfill mission objectives [1]. Therefore, essential services must survive even when the system is infiltrated, compromised or crashed. Survivability strategies, *resistance, recognition* and *recovery* (RRR), are in this case cornerstone system capabilities. In particular, resistance is the capability to repeal attacks and to mask faults, recognition is the capability to detect attacks and faults and to evaluate damage and, recovery is the capability to restore services after intrusions or failures.

We defend that an early evaluation of the survivability properties of a critical system may ensure a system deployment that better guarantees resistance to faults or attacks, i.e., the system specification helps to prove that essential services can be offered timely, despite the presence of faults or attacks. In the previous work [2] we proposed a model-driven approach to assess survivability requirements using Petri nets [3] that is compliant with the Survivable Network Analysis (SNA) [1]. The language used for specification is UML [4] and two UML profiles: misuse cases [5] and DAM [6]. In particular, misuse cases are used to specify faults and attacks threatening essential services. Besides, the DAM profile enables to define the QoS for the essential services and the survivability requirements against the identified faults and attacks. As a byproduct of the misuse case and DAM specification, our method yields a *service modes specification*, namely SAM (survivability assessment model). A SAM comprises full service and degraded modes and paths that account for threats and survivability strategies. Special feature of the method is its ability to verify whether fundamental properties for survivable systems are considered in the requirements specification, such as the capability of the system to recover the full service mode. The method then provides a support for the engineer to identify lacks or errors in the SAM. The method primarily focuses on how to represent the system requirements for getting an appropriate SAM, rather than addressing the issues of eliciting and gathering the requirements, which definitely are in the use case technique [7]. Model-driven techniques are used to derive from the SAM a Petri net model, where model-checking is applied for a system survivability verification.

This paper extends the initial contribution in [2] as follows: 1) we include the background needed to understand our proposal and 2) supplement the related work by considering the relevant tools that support the assessment of non functional properties, specified with high level modelling languages such as UML and AADL, through model checking. 3) We also present a tool -*MASDES*- that automatizes almost all the steps of the proposed method. In particular, the MASDES tool has been developed within the Eclipse Workbench [8] and relies on Papyrus tool [9] for the UML modelling and profiling features. It consists of a set of plug-ins that enable to create a survivability system view using UML and profiling techniques, and to verify survivability properties. In particular, the tool performs model transformations in two steps. First, a model-to-model transformation generates from the survivability view a Petri net model in a tool-independent format. The latter is defined by an *e-Core* meta-model which is compliant to the standard PNML [10, 11] -for what concerns the definition of the Petri Net structure- and enhances it to encompass the properties to be checked. In the second step, model-to-text transformations produce Petri net specifications for a specific model-checker.

The paper is organized as follows. Section 2 reviews the related work. Section 3 briefly summarizes the SNA method and the languages (DAM, use cases and Petri nets) needed to understand our proposal. Section 4 describes our method. Section 5 describes the architecture and main functionalities of the tool that supports the method. Section 6 illustrates the method using a military C2IS case study. Conclusions are drawn in Section 7.

2. Related Work

Survivability has always been an important requirement in the military context for platforms, communication systems, and nowadays more generally to missions. It has also been a concern of those civil system domains (e.g., information systems and critical infrastructures) where it is crucial to guarantee certain QoS levels despite a set of pre-specified threats.

In the military context, Knight and Strunk [12] proposed a formal definition for acceptable levels of service offered by a survivable system under different environment conditions. Each level is quantified by relative values (as perceived by the users) and can be expressed in terms of QoS requirements. The survivability specification is a graph, where the nodes

represent the acceptable levels of service offered by the system and the edges model the change of levels when certain environment conditions are met. We have also used a graph representation for the different system service modes. Our work goes one step further by supporting the formal verification of the survivability properties.

Several other approaches have been proposed in the literature to collect QoS requirements related to survivability. Mustafiz et al. [13] define a requirement engineering process to elicit reliability/safety requirements and degraded services. Similar to our proposal, use cases are profiled to model undesired situations that can interrupt the normal behavior of the system and handlers to guarantee reliable and safe services. In [14], a framework (UMD) has been proposed for eliciting and modelling dependability requirements that is designed around a basic modelling language defined by the authors. As in our proposal, UMD can be used to identify and define measurable dependability requirements and properties of the system. Allenby and Kelly [15] integrate use case and hazard identification techniques for safety requirements elicitation in aerospace application domain. The work in [16] inherits from [15] the use case specification and applies Practical Formal Specification to specify safety requirements and to verify them for completeness and consistency.

Regarding tools devoted to systems survivability assessment, we have not found in the literature any relevant option. However, there are some related works that deserve to be mentioned. TABU [17] is a tool for automatic verification of UML Class, State Machines and Activity diagrams through model checking. TABU automatically transforms UML specifications in labeled transition systems that are then analyzed using the SMV (Symbolic Model Verifier) tool. PRIDE tool-set [18] provides support for dependability evaluation of complex systems, from qualitative and quantitative point of views. In particular, concerning qualitative analysis, it enables to verify the correctness of UML State Machines through model checking. Similarly to TABU, the target formal model is a labeled transition system that, in case of PRIDE, is expressed as a PROMELA model, the modelling language used by the SPIN model checker. COMPASS tool-set [19] consists of several tool components: NuSMV, RAT, FSAP or MRMC. COMPASS relies on probabilistic model-checking for performability analysis and on model-checking for requirements and safety analysis as well as for diagnosability. At modelling level, COMPASS uses AADL (Architecture Analysis and Design Language) instead of UML. Last but not least, INVEST [20] is a framework for reverification of component-based software systems after changes (such as additions, removals or modifications of components), so it is appropriate for systems whose structure changes dynamically. INVEST verifies probabilistic safety properties using PRISM [21] model-checker.

3. Background

3.1. Survivable Network Analysis

The Software Engineering Institute at Carnegie Mellon University has developed technologies for survivability analysis of highly distributed, unbounded network environments with no central control or unified security policy. In particular, they proposed the Survivable Network Analysis (SNA) method [1] and apply it to the analysis of failures and accidents as well. The SNA method focuses on systematically designing survivability into systems during development and evolution. A small team of evaluators, typically three or four people, apply the method to an existing proposed system. Evaluators must be knowledgeable in a number of disciplines, including architecture analysis, intrusion techniques, and survivability strategies.

The SNA method proposes four steps. In Step 1, the team reviews mission objectives and system requirements, moreover they elicit the system architecture. In Step 2, the team identifies essential services and assets, based on mission objectives and the consequences of failure. Usage scenarios characterize essential service and asset uses. These scenarios are mapped onto the architecture as execution traces to identify the composition of corresponding essential components, which must be available to deliver essential services and maintain essential assets. In Step 3, the team selects intrusion scenarios based on the system environment and an assessment of risks and intruder capabilities. These scenarios are likewise mapped onto the architecture as execution traces to identify corresponding compositions of compromisable components, or components that

intrusion could penetrate and damage. In Step 4, the team identifies the architecture's soft-spot components as components that are both essential and compromisable, based on the results of Steps 2 and 3.

The method proposed in this work complies with the aforementioned SNA steps 1, 2 and 3 for what concerns the activities related to requirement analysis.

3.2. Dependability Analysis and Modeling profile

The "Dependability and Analysis Modeling" (DAM) profile [22] is a specialization of the Object Management Group (OMG) standard profile "Modeling and Analysis of Real Time and Embedded systems" (MARTE) [23]. DAM enables to specify dependability requirements according to the well-defined Value Specification Language (VSL) syntax, which is inherited from MARTE.

In this work, we propose DAM within a method that helps engineers to capture survivability requirements -focusing on availability, performance, confidentiality and integrity aspects- and to assess the specification of the acceptable service modes. A DAM annotation *stereotypes* the model element it affects in the way UML proposes, i.e., by extending its semantics. The UML diagrams used in our approach are *misuse case* and *state machine* diagrams, then the DAM extensions will be applied to (mis)use cases in use case diagrams -see Figures 2(A), 6 and 8- and states/transitions in state machines -see Figures 2(B1,B2) and 7. The DAM stereotypes used in this paper are summarized in Table 1.

Misuse cases diagram		
Stereotype	***UML elements (description)***	***Tags (description)***
daService	use cases (essential services)	
daStep	misuse cases (threat scenarios)	kind, fault (fault characterization in terms of objective, persistence, effect -domain, detectability, consistency, consequence-, and affected Non Functional Requirements -NFR)
daResistance, daRecognition, daRecovery	use cases (survival strategies)	servMode (service mode achieved with the strategy)
State machines diagram		
Stereotype	***UML elements (description)***	***Tags (description)***
daServiceMode	state	service (referenced essential services) qos (set of NFR)
daChange	transition	threats (referenced misuse case) resistance, recognition, recovery (referenced use cases)

Table 1. Subset of DAM extensions used in the methodology.

According to UML, each DAM stereotype is made of a set of tags which define its properties. For example, *daStep* stereotype has *kind* and *fault* as tags. The former is used to specify the kind of step (e.g., "fault") and the latter enables to characterize the nature of the fault, such as its origin -"accidental" or "malicious"- and its persistence -"transient" or "persistent". The types of the tags are basic UML types, enumeration types -such as the type of the *kind* tag-, MARTE NFP types or complex dependability types -such as the type of the *fault* tag. The latter are composed of attributes which may be MARTE NFP types or simple types.

MARTE NFP types are data-types of special importance since they enable the description of relevant aspects of the NFR 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).

3.3. Use Cases and Misuse Cases

Use cases [24] is a very well-known technique for eliciting requirements and likely the most popular. 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, in particular survivability requirements. To address this lack, *misuse cases* [25] were proposed for eliciting mainly security and safety requirements.

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 2, use cases and actors can coexist with misuse cases and hostile actors (the latter are shown in bold in the figure). Once the analyst identifies a misuse case *threatening* the system, such as "MUC1", then s/he has to act to *mitigate* its effects. Usually, this takes the form of a new use case, e.g., "UC1-1". Again, the hostile actors may find the way to threaten the system. 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.

3.4. Petri Nets

A Petri net (PN) [3] is a bipartite graph, in which the vertices can be either transitions or places. The transitions, graphically depicted by bars, represent events that may occur in the system; the places, represented by circles, are used to model conditions. The directed arcs, shown by arrows, describe which places are pre- or post-conditions for which transitions. Places may contain tokens, depicted by black dots; the (initial) distribution of tokens over the places of a PN is called (initial) marking. An exemplification of its graphical representation is shown in Figure 3(b).

The PN dynamics is governed by the transition enabling and firing rules. A transition is enabled whenever there is at least a token in each of its pre-condition places, and it may fires if there are not enabled transitions with higher priority. When it fires, a token is consumed from each of its pre-condition places and a token is produced in each of its post-condition places. A reachable marking is then a marking reached through the firing of a transition sequence from the initial one. The reachability graph associated to a PN represents the reachable markings (vertices) and the transition firings causing the change of markings (edges).

Petri Nets have been used to verify the correctness of the modelled system in terms of *desirable properties*, which are defined considering place markings and/or transition firings [26]. The *boundedness* of a PN characterizes the finiteness of its state space (i.e., the set of reachable markings): every place of the PN is bounded and the bound is finite. The *fireability* is related to the potential firing of the PN transitions: a transition t is fireable, at a given marking m, if there exists a transition firing sequence leading to a marking m' in which the transition t is enabled. *Dead-lock* freeness is also related to the fireability of the PN transitions: a PN is dead-lock free if always -i.e., from any reachable marking- at least one transition can fire. The *liveness* property is stronger than fireability: indeed, a transition t is live if it never loses the possibility of firing (i.e., it is fireable in all reachable markings) and the PN is live if all its transitions are live. The *recoverability* (home state) is the property of a marking to be reachable from any other reachable marking. The *live-lock* freeness is related to the recoverability of the initial marking since a PN is live-lock free if there are not terminal live behaviours. Finally *causality* and *mutex* (i.e., mutual exclusion) are properties that consider the firing of two transitions: transitions t and t' are causally related if the firing of t eventually causes the firing of t', whereas transitions t and t' are mutually exclusive if they cannot fire simultaneously.

4. A Method for Survivability Assessment

The system survivability assessment, herein proposed, is conceived as a *method* inside the requirements analysis stage of the software life-cycle, see Figure 1. The method has four steps, being the first two compliant with the SNA method previously described. The goal is to obtain a requirements specification that accounts for system survivability requirements, that need to be verified formally. The steps of the method need modelling and analysis tool support, as indicated by the

legend in Figure 1. From the software engineering point of view, the method can be used within iterative and incremental processes, e.g. the Rational unified process [27], as a step that complements the Requirements analysis stage.

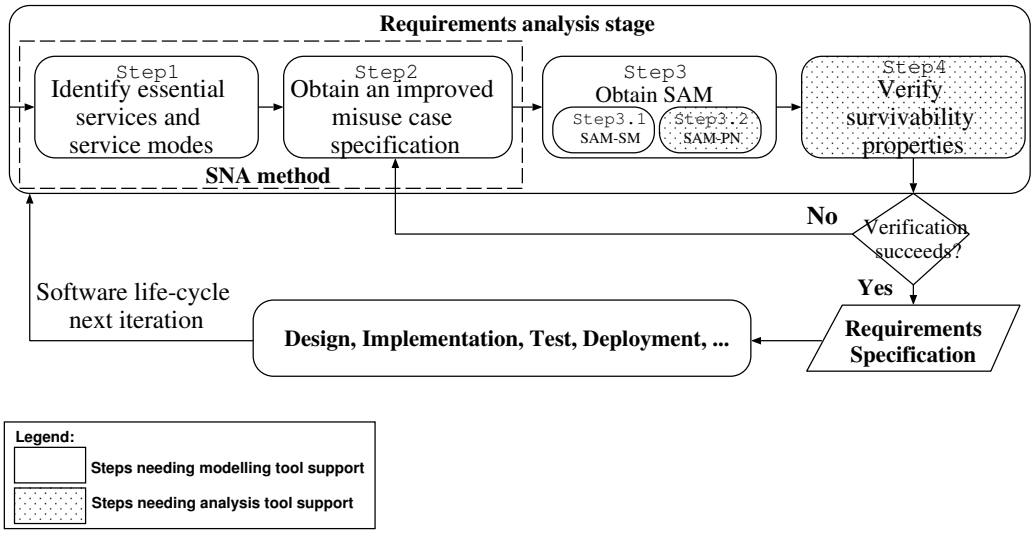

Fig. 1. Software life-cycle: emphasis on survivability assessment.

4.1. Identify Essential Services and Service Modes

From the use cases of the system, the engineer selects those that must not collapse, even when the system operates in adverse conditions as fault or attack. Our process supplements the use case specification as follows:

- Each selected use case is considered an essential service and it is stereotyped as `daService`.
- The engineer develops a QoS specification for each essential service. The QoS of an essential service is defined by assigning, for each service mode, acceptable threshold values to a subset of measures of interest (see for example Table 5). Then, firstly, s/he identifies the system service modes the system should ensure (e.g., full functionality, degraded performance and degraded confidentiality). Secondly, s/he identifies measures (e.g., availability, response time) and indicators (e.g., confidence level) of interest and finally establishes -for each service mode- the acceptable thresholds for measures and indicators.

Discussion. The service modes are system-specific and their identification should be carried out with help of domain experts. For this reason, our approach does not provide any particular criterion for their identification. However, the set of service modes usually include a service mode that represents the best QoS offered by the system for all the identified essential services, namely the *full functionality* mode. The other service modes represent instead the QoS degradation of the essential services [12]. Regarding the number of service modes it is expected to have a low number; three or four different services modes can be reasonable even for large systems. Service modes in Table 5 describe a paradigmatic case.

4.2. Obtain an Improved Misuse Case Specification

The objective is to develop an improved misuse case specification, which consists of both, a specification of the threats to the system and a set of survivability strategies - resistance, recognition and recovery -.

Step 2.1. Threats specification Here we propose a faults and attacks specification that supplements the misuse case technique. Fault and attack scenarios *threatening* the essential services are discovered and represented by misuse cases (i.e.,

use cases stereotyped as `misuse` and `daStep`). Similarly to use cases, also misuse cases are detailed with descriptions, Table 6 shows an example. Each misuse case is annotated with `daStep` tagged-values that describe the characteristics of the threat, according to the classification in [28]. The engineer should, at least, specify the *persistency* of the threat (i.e., transient or permanent), the *origin* (i.e., malicious, if the threat is an attack, or not malicious if the threat is a fault), the *effect* on the threatened essential services, that is the service failure modes, and the affected NFR (i.e., the affected measures/indicators defined in the QoS specification). In particular, the service failure modes characterize the incorrect service according to different viewpoints: the *domain* (i.e., content, early or late timing, halt, erratic), the *detectability* (i.e., signaled or unsignaled), the *consistency* (i.e., consistent or inconsistent) and the *consequence* on the environment (application-specific severity levels are normally used which are associated to maximum acceptable probabilities of occurrence). The misuse case annotations in Figures 6 and 8 give examples of the threats specification.

Step 2.2. Survivability strategies For each misuse case, survivability strategies need to be specified to *mitigate* the misuse case effects on the essential services. The suggested order for identifying the survivability strategies, i.e., first resistance, then recognition and finally recovery, follows the SNA method [1]. Each strategy can be interpreted as a new service added to the system to increase the survivability of the essential services, then represented as a new use case. A strategy is stereotyped either `daResistance`, `daRecognition` or `daRecovery` according to the classification in [1]. The survivability strategies mitigating a given misuse case are identified as follows:

1. The engineer studies whether one or more *resistance* strategies can be devised, and creates a use case per strategy. The interpretation is that the resistance introduced by *UC1-1* AND *UC1-2*, see Figure 2, when applied to *MUC1* would leave the system in full functionality. However, it is also possible that the resistance does not succeed, then the system reaches a degraded service mode that is specified as a tagged-value associated to the UCs (*servMode* in Figure 2(A)).
2. The engineer studies whether a strategy might *recognize* the degraded service mode induced by the success of the misuse case. If the engineer identifies such strategy, s/he creates the corresponding use case (*UC1-3* in Figure 2(A)).
3. The engineer studies strategies to *recover* the system for those cases that the resistance does not success, and s/he creates one use case for each recovery strategy. Each use case is annotated with a tagged value indicating the impact of the strategy. For example, one strategy could recover the system to the full functionality, but other could get less impact (see Figure 2(A), *UC1-5* and *UC1-4* respectively).

Sub-step 2.3. Review the QoS specification The improved misuse case specification -created by Steps 2.1 and 2.2- will help the engineer to review the QoS specification initially proposed in the first step of the method. So, new service modes can be added, new metrics can be devised, and modifications in the thresholds introduced. Again a few experience in QoS is required.

Discussion. With respect to the threats specification -Step 2.1- we have not considered possible dependency relations between misuse cases. In principle, the UML standard relations «`include`» and «`extend`» could be exploited for this purpose. Nevertheless, this is an open issue that deserve to be investigated to improve our proposal in the light of the current sophisticated forms of cyberattacks, like APT [29], which increasingly often threat ICT-based critical infrastructures.

4.3. Obtain a Survivability Assessment Model

Two equivalent survivability assessment models (SAM) are built through two sequential steps. First, a SAM state machine (SAM-SM) is obtained by leveraging the improved misuse case specification, obtained in Step 2, and QoS specification, obtained in Step1. The SAM-SM provides to the engineer a complementary view of the misuse case specification by explicitly showing the system service modes and their change due to threats.

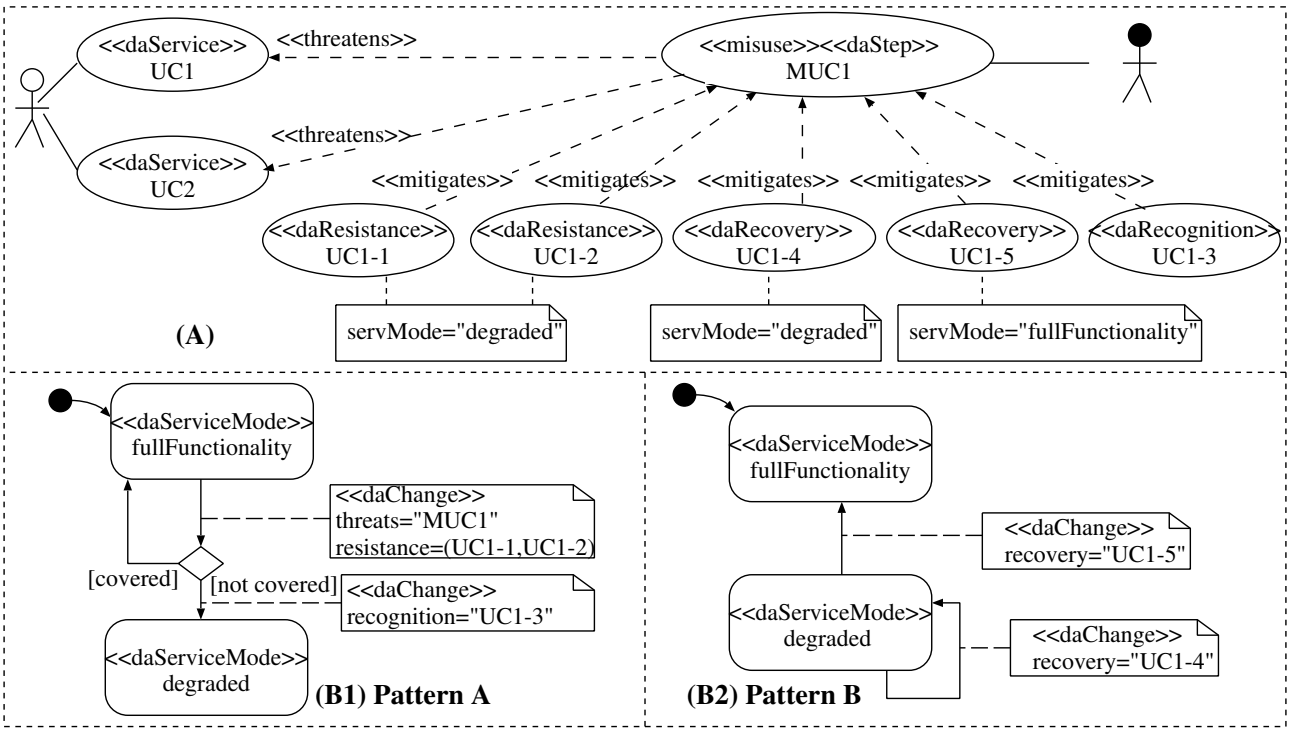

Fig. 2. Service modes specification patterns.

The SAM-SM is a UML state-machine whose states are the system service modes in the QoS specification. The transitions between states are obtained from the improved misuse case specification. Our tool does not currently support the automatic generation of the SAM-SM model, however the algorithm can be easily implemented as informally described in the following.

We consider two patterns to obtain the transitions (see Figure 2). Starting from the *fullFunctionality* service mode, for each misuse case we first apply pattern A and then pattern B:

- *Pattern A* (Figure 2-B1). We create a choice node[1] whose input is the full functionality state and has two output transitions. One to the full functionality mode to represent that the resistance to the considered misuse case succeeds. The other output leads to the degraded mode specified by the *servMode* tagged-value associated to the resistance strategies (Figure 2-A). The transition from the full functionality state to the choice node is labelled `daChange` and the tagged-values specify the mitigated misuse case (*threats*) as well as its resistance strategy use cases. The `daChange` transition from the choice node to the degraded state specifies the recognition use case.
- *Pattern B* (Figure 2-B2). We review the recovery strategies for the considered misuse case. For each one, we create a transition whose input is the degraded mode induced by the misuse case and its output the target mode indicated by the strategy (*servMode* tagged-value). The transition is labelled `daChange` and indicates the recovery use case.

The SAM-SM is still a UML-based specification so it is not directly analyzable with formal methods (e.g., via model checking). Then, a further step is required in order to get a formal model from the SAM-SM obtained as described previously. Our proposal is a Petri Net model, namely SAM-PN, that is automatically derived from the SAM-SM by applying a model-to-model transformation. Figure 3 sketches the mapping: the SAM-SM on the left is actually the one produced in the Section 6 -the C2IS case study- for the first iteration (cf. Figure 7, white part). The translation approach is quite intuitive: SAM-SM states are mapped to single PN places, while SAM-SM transitions correspond to sequences of causally connected

[1] The choice node is graphically represented by the diamond shape.

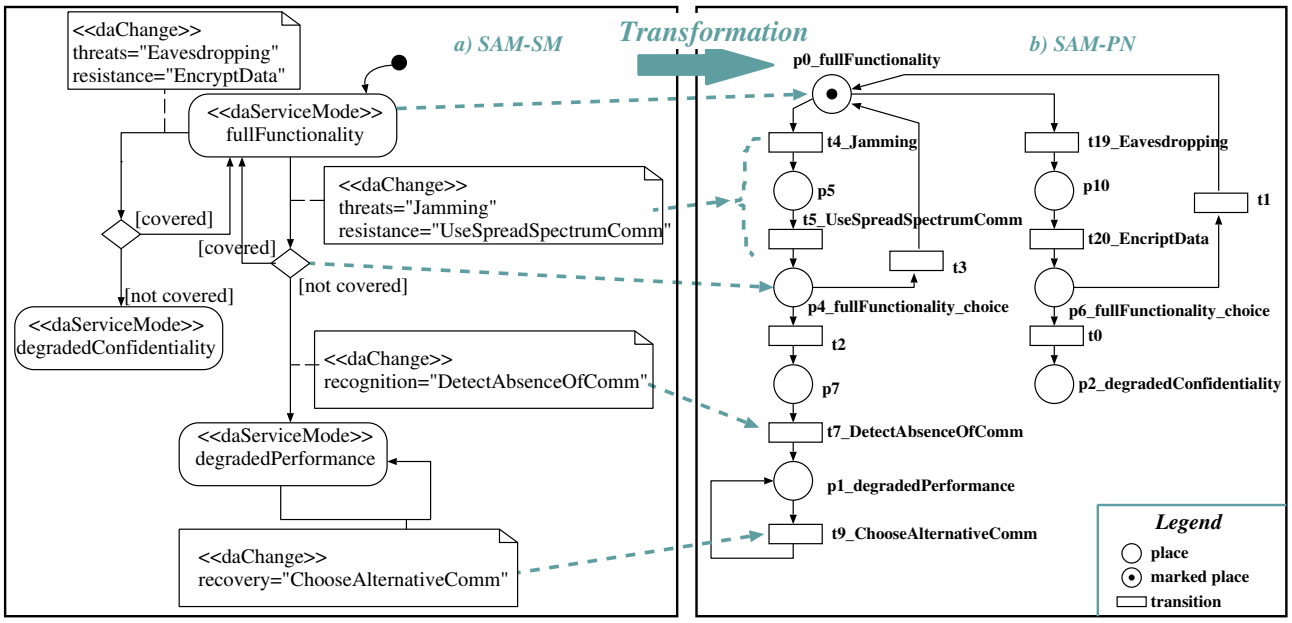

Fig. 3. Model transformation.

PN transitions where the number of the latter depends on the annotations associated to the former. In particular, a change of service mode can be characterized by a threat causing it (*threats* tagged-value) and/or by a survivability strategy aimed at mitigating it (*resistance*, *recognition*, *recovery*). So each tagged-value, annotated to a `daChange` SAM-SM transition, is mapped to a PN transition, where the name of the transition consists of the transition identifier concatenated with the tagged-value. For example, the transition *t4_Jamming* results from the mapping of the *threats* tagged value associated to the SAM-SM transition between the *fullFunctionality* and choice node on the right. The causality of the threat occurrence and the consequent resistance and/or recognition and/or recovery is captured by the causal connection of the corresponding PN transitions. The choice in the SAM-SM is translated to a free-choice conflict between two PN transitions: one representing the successful coverage of the resistance strategy (e.g., *t3*) and the other the unsuccessful case (e.g., *t2*). Finally there is a unique marked place (initial marking), which is the one corresponding to the initial state of the SAM-SM.

Discussion. The following two assumptions have been made to get the SAM-SM model: first, we assume that there exists an initial service mode where the essential services are fully operational, such as the *fullFunctionality* mode in Figure 2(B1,B2). As already discussed in *Step 1*, this is not a limitation for the approach. Second, our proposal relies on the *one-threat assumption*, that is starting from the initial service mode the model represents the sequence of events/actions triggered by a threat occurrence: no further degradation is considered once a degraded service mode is reached. Then, the complexity of the algorithm that generates the SAM-SM model from the misuse case and QoS specifications takes linear time $O(n)$, where n is the number of misuse cases. This restriction can be relaxed by considering all the possible combinations of threats, then enabling a more extensive analysis support in the following *Step 4*. However, this enhancement comes at a price of the model comprehensibility and of the algorithm complexity ($O(n!)$). A challenge is to find a good trade-off between fault assumption, model comprehensibility and algorithm complexity.

4.4. Verify Survivability Properties

In the last step, the SAM-PN is used for verifying survivability properties through model-checking techniques. The step aims at providing a feedback to the engineer about the completeness and correctness of the service modes specification. In particular, considering that the SAM represents the acceptable service modes of the essential services and the change of

service modes due to adverse conditions (attacks or faults), it may be interesting to check, at least, the following properties of survivability:

P1. The system should always be able to recover to the full functionalities.

P2. The survivability strategy S is feasible.

P3. As a response to the occurrence of an adverse condition C, the system should be able to carry out the survivability strategy S.

Model checking techniques [30] are state space based techniques that consist in verifying logical properties on the reachability graph of the PN model. Logical properties need to be formally specified as queries in order to be processed by a model checker: there exist different temporal logic languages that can be applied (e.g., CTL [31], LTL [32]), depending on the type of property to be checked and on the type of model checking technique. On the other hand, we need to interpret the survivability properties in terms of logical properties of the PN model, which will be then expressed as queries on the PN reachability graph using a formal language.

	PN property	PN query	Logical condition description
P1	Home state	Q1: AG(EF($init$))	initial PN marking ($init$)
P2	Fireability	Q2: EF(pre_s)	enabling set of s (pre_s)
P3	Causality	Q3: G(pre_c ⇒F pre_s)	enabling sets of c and s (pre_c, pre_s)

Table 2. Mapping of properties to PN queries.

Table 2 shows the PN properties (second column) that ensure the satisfiability of the properties **P1-P3** and the PN queries (third column) that can be formulated using temporal logic languages. All the queries are characterized by logical conditions on the PN marking (fourth column).

In particular, the full functionality service mode in the SAM-SM is represented by the initial marking of the PN model. The recoverability of the former (**P1**) is ensured if the latter is a *home state*, that is if it is reachable from any other reachable marking. The feasibility of a survivability strategy (**P2**) corresponds to the *fireability* of the PN transition s representing the strategy, that is the latter belongs at least to a firing sequence. Finally, the cause-effect relationship between the occurrence of an adverse condition C and the execution of a survivability strategy S (**P3**) corresponds to the *causal dependence* between the PN transitions c and s modelling C and S, respectively. Observe that, for a given PN property, different possible queries can be formulated; in Table 2, Q1 and Q2 are CTL formulas and Q3 is an LTL formula.

Once the PN queries have been defined, they can be executed using a PN model checker: besides the true/false answer, usually the model checker produces a counter-example path for queries of universal type (e.g., Q1 and Q3) and a witness path for queries of existential type (e.g., Q2). For example, using PROD [33], the query Q1 on the PN model of Figure 3(b) returns a false value. A counter-example path is also produced that indicates a path, on the reachability graph of the PN model, leading to a deadlock marking (i.e., *p2_degradedConfidentiality* place marked).

Discussion. The three PN queries -listed in table 2- are quite simple temporal logic properties, but indeed they are crucial from the survivability requirement assessment point of view. Other interesting logical properties could be added to the list such as: 1) the precedence, to check the order of execution of the RRR strategies as a response of a given threat occurrence, and 2) the mutual exclusion, to check that different recovery strategies cannot be carried out at the same time to mitigate a given threat occurrence.

5. Tool Architecture and Main Functionalities

The method for survivability requirements assessment, described in the Section 4, is supported by a software tool called *MASDES* [34]. The MASDES tool has been developed within the Eclipse Workbench [8] and relies on Papyrus [9] tool for

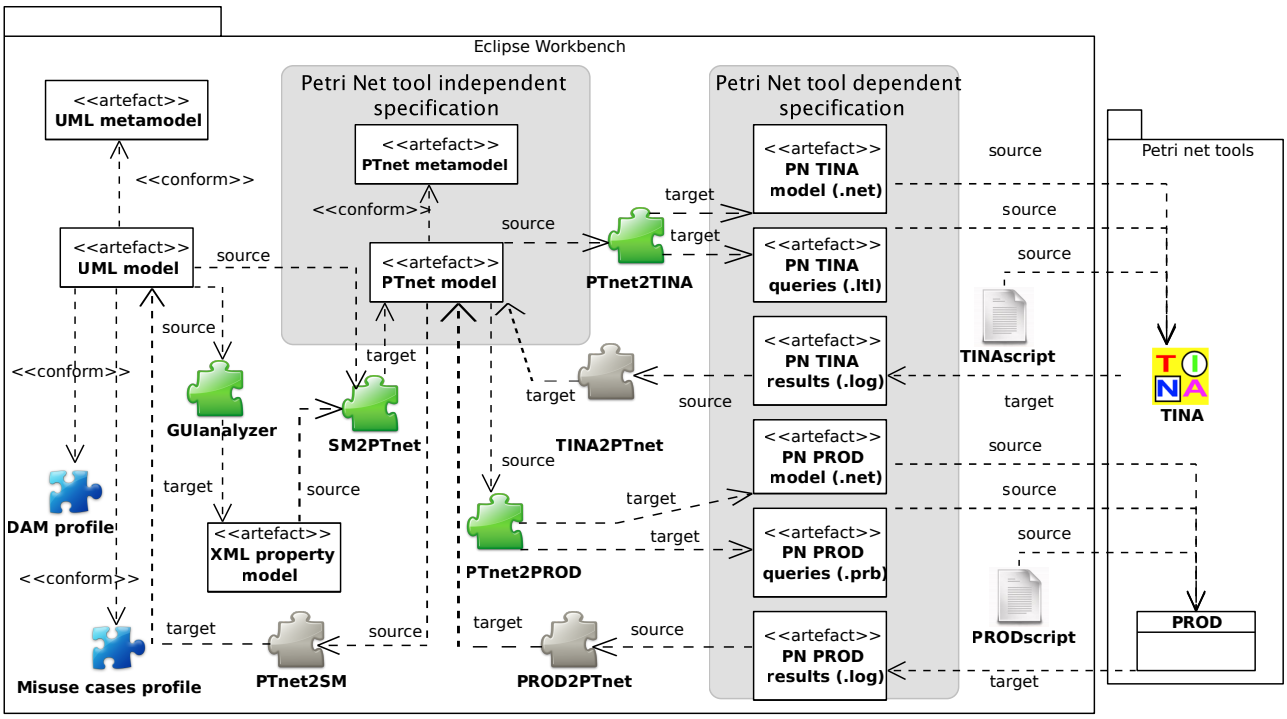

Fig. 4. Components, artefacts and information flow.

the UML modelling and profiling features. Figure 4 provides an overview of the tool, with the main components (puzzle's pieces), the conformance of the UML and PN models to the corresponding meta-models (*conform* relations), the artefacts either used or generated by the components (*source* and *target* relations, respectively), and the information flow (inferred by the direction of the *source* and *target* relations). The Petri net tools (right-hand side of the figure) are external tools used for analysis with model-checkers.

In the following, Subsection 5.1 describes the main components of the tool, Subsection 5.2 briefly reviews the modelling support provided by the tool, and Subsection 5.3 details the analysis functionalities.

5.1. Tool components

MASDES supports the modelling and analysis of survivability specifications. On the modelling side, UML profiles -misuse cases and DAM- have been developed as separate Eclipse plug-ins, then enabling their independent usage. On the analysis side, model-2-model (M2M) and model-2-text (M2T) forward transformations have been developed to automatically get:

1. A Petri net model from UML-based specifications and
2. Queries to be submitted to Petri net model checkers, which are submitted by the user via a GUI (*GUIanalyzer*).

All the analysis related components (i.e., GUI, transformers, scripts to launch the model-checkers) have been wrapped up in an Eclipse plug-in. The transformations, implemented in the Atlas Transformation Language (ATL) [35], are carried out in two steps. First, a M2M transformation generates a Petri net model and property specification, both in a tool-independent format (*SM2PTnet* component). Second, M2T transformations produce a Petri net model and queries, which are expressed in either temporal logic formulas or Petri net tool-specific commands. Currently, two M2T forward transformations have been implemented, toward TINA [36] (*PTnet2TINA*) and PROD [33] (*PTnet2PROD*) model checkers. Backward transformations, that provide the feedback of the analysis to the original UML-based specification (*TINA2PTnet, PROD2PTnet* and *PTnet2SM*), are still not available but planned as future work.

5.2. Modelling support

MASDES provides modelling support to the methodology described in Section 4 by implementing those features needed to accomplish Step1, Step2 and Step3.1 (see Figure 1). In particular, an improved misuse case specification can be obtained by applying UML profiling techniques, that is by using misuse case and DAM profiles.

It is worth to notice that, although the methodology actually imposes the use of only a subset of the DAM profile extensions (summarized in Table 1), both profiles are fully supported by the tool and can be used independently from the methodology and the tool analysis features discussed in the following subsection.

5.3. Analysis support

MASDES provides analysis support to the methodology by implementing Step3.2 and Step4 (see Figure 1). Herein, the information flow produced during an analysis session is detailed (see Figure 4).

GUI analyzer. An analysis session starts by selecting via the *GUIanalyzer*: 1) the Survivability Assessment Model State Machine (SAM-SM), 2) the properties to be checked, and 3) the Petri net tool used as model-checker (i.e., either TINA or PROD). In particular, the properties to be checked (e.g., **P1-P3** defined in Subsection 4.4) are related to the service modes and the changes of service modes specified in the selected SAM-SM, due to the threat occurrences and the survivability strategies defined in the misuse case diagram. Such properties are shown as parameterized templates, expressed as English sentences, which are instantiated when the user selects the model elements (i.e., service modes, misuse, survivability strategies) of interest from the GUI combo-boxes, as follows:

PT1. The system should always recover to $< serviceMode >$ (*recoverability*).

PT2. The strategy $< survivabilityStrategy >$ is feasible (*strategy feasibility*).

PT3. As response to the threat $< misuse >$, the system should be able to carry out the strategy $< survivabilityStrategy >$ (*threat mitigation*).

M2M transformation. Both, the UML-based specification (*UML model* artefact) and the properties instantiated by the user through the GUI (*XML property model*) are the source artefact models of the M2M transformer *SM2PTnet*, which has been implemented with ATL using the “matching rules" paradigm. The M2M transformation rules implement the mapping of Figure 3.

SM2PTnet produces a tool-independent specification of the Petri net model and properties compliant to the *PTnet* meta-model (see Figure 5), which is an e-Core model defined with the Eclipse Modelling Framework. Petri net basic concepts (white meta-classes) conform to the ISO/IEC standard PNML [10, 11]. Then, a Petri net document (`PTnetDoc` meta-class) may contain one or more Petri nets (`PTnet`). Each Petri net consists of a set of *nodes*, either *places* or *transitions*, and a set of *arcs*. Places are characterized by an initial marking (`ptInitMarking` meta-attribute), arcs can be either normal or read arcs (`kind`) and have a multiplicity, and transitions are characterized by a minimum and maximum firing delay (`minTime,maxTime`). Observe that the transition timing attributes are not part of the PNML standard, but they are defined in the PNML implemented by the TINA tool. However, we have decided to keep them in the meta-model for future extensions of the tool which include the analysis of timing-dependent properties.

The *PTnet* meta-model also includes concepts that represent well known logical properties to be checked on the Petri net as well as the counter- or witness- examples produced by most of the Petri net model checkers (blue meta-classes).

A set of *logical properties* is then associated to a Petri net, each one with a boolean value indicating whether the property is satisfied by the Petri net (by default the value is set to `false`). Different types of properties are considered: *boundedness*

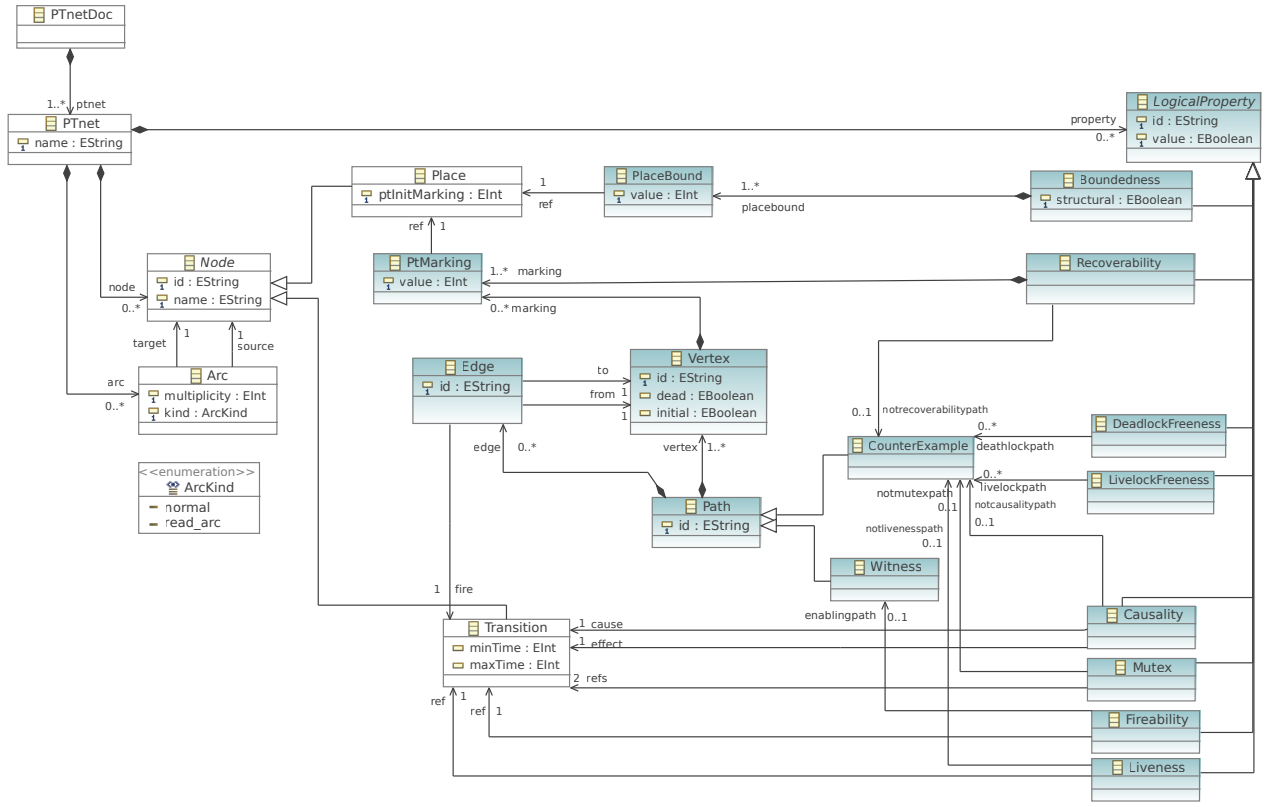

Fig. 5. PTnet meta-model.

of places with the associated bound values (`PlaceBound`), *recoverability* of a given marking, *dead-lock* and *live-lock freeness*, *causality* and *mutual exclusion* of two transitions, transition *fireability* (i.e., 0-liveness) and *liveness*.

All the properties but the place boundedness are related with either *counter examples* or *witness* examples, that are *paths* on the reachability graph associated to the Petri net. Each path consists of a set of *vertices*, that is the marking reachable from the initial one, and a set of *edges*, that represent transition firings.

To the best of our knowledge, no meta-model or XML-based language has been proposed in the literature for the specification of the Petri net properties: so this work also provides a contribution toward the definition of a standard.

Table 3 shows an excerpt of the *PTnet model* generated by the M2M transformation, which corresponds to the SAM-SM of Figure 3(a). In particular, the `property` tags correspond to the Petri net logical properties produced when the user selects the following properties in the GUI:

PI1. The recoverability of the *fullFunctionality* service mode (cf. Figure 7),

PI2. The feasibility of the *ChooseAlternativeComm* strategy (cf. Figure 6),

PI3. The mitigation of the *Jamming* threat with the *UseSpreadSpectrumComm* resistance strategy (cf. Figure 6).

Observe that the above properties are instantiation of the property templates **PT1**…**PT3**.

M2T transformations. The M2T transformations generate tool-dependent Petri net models and queries from the PNnet model specification produced by the *SM2PTnet* transformer. Currently, two M2T transformations are supported by MASDES, both implemented in ATL using the “query" paradigm: *PTnet2TINA* and *PTnet2PROD*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PTnet:PTnetDoc xmi:version="2.0"
                xmlns:xmi="http://www.omg.org/XMI"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:PTnet="https://bitbucket.org/.../PTnet">
 <ptnet name="SAM-Itl">
  <node xsi:type="PTnet:Place" id="p0" name="fullFunctionality" ptInitMarking="1"/>
  <node xsi:type="PTnet:Place" id="p2" name="degradedConfidentiality"/>
  <node xsi:type="PTnet:Place" id="p1" name="degradedPerformance"/>
  <node xsi:type="PTnet:Place" id="p6" name="fullFunctionality_choice"/>
  <node xsi:type="PTnet:Place" id="p4" name="fullFunctionality_choice"/>
...
  <node xsi:type="PTnet:Transition" id="t19" name="Eavesdropping" maxTime="-1"/>
  <node xsi:type="PTnet:Transition" id="t9"  name="ChooseAlternativeComm" maxTime="-1"/>
  <node xsi:type="PTnet:Transition" id="t20" name="EncryptData" maxTime="-1"/>
  <node xsi:type="PTnet:Transition" id="t4"  name="Jamming" maxTime="-1"/>
  <node xsi:type="PTnet:Transition" id="t5"  name="UseSpreadSpetrumComm" maxTime="-1"/>
  <node xsi:type="PTnet:Transition" id="t7"  name="DetectAbsenceOfComm" maxTime="-1"/>
...
  <arc target="p1" source="t2" multiplicity="1"/>
  <arc target="p1" source="t9" multiplicity="1"/>
...
  <property xsi:type="PTnet:Recoverability" id="lp3">
   <marking value="1" ref="p0"/>
  </property>
  <property xsi:type="PTnet:Fireability" id="lp1" ref="t9"/>
  <property xsi:type="PTnet:Causality" id="lp0" cause="t4" effect="t5"/>
 </ptnet>
</PTnet:PTnetDoc>
```

Table 3. PTnet model generated by the *SM2PTnet* transformer.

The former produces two files (i.e., `.net` and `.ltl`) that describe, respectively, the Petri net model and the queries in the syntax of TINA [36]. In particular, the queries are expressed as LTL [32] formulas or as TINA specific formulas. Similarly, *PTnet2PROD* produces two files (i.e., `.net` and `.prb`) that describe, respectively, the Petri net model and the queries in the syntax of PROD [33]. In this case, the queries are expressed as either as CTL [31] formulas or as PROD specific formulas.

Figure 3(b) shows the graphical representation of the PN model that has been generated, by the *PTnet2TINA* transformer, from the PTnet specification in Table 3. The queries produced in PROD and TINA are shown in Table 4: the recoverability and fireability are properties expressed as CTL formulas, then their corresponding queries are only generated for PROD. The causality property is expressed as an LTL formula, then the corresponding query is generated only for TINA.

```
/* Queries: SAM_Itl */

/* Recoverability */
check ag(ef((p0_fullFunctionality == <.1.>) and true))

/* Fireability */
check ef((p1_degradedPerformance >= <.1.>) and true)
```

```
# Queries: SAM_Itl

#  Causality
[] (t4_Jamming=> (<>t5_UseSpreadSpetrumComm) );
```

Table 4. Petri Net queries in PROD (top) and TINA (bottom).

Finally, the Petri net tool model checker selected in the GUI is launched by a proper shell script (*TINAscript* or *PRODscript*). The results, produced by the model checker in a textual format, are stored in a log file.

6. The C2IS Case Study

Similarly to Knight and Strunk [12], we focused on the military context to choose a case study in order to illustrate the use of the method proposed here and the software tool developed to support it. Specifically, we chose a military Command and Control Information System (C2IS) [37] as case study. Generally, the C2IS systems share information to synchronize the *Situational Awareness* and the *Purpose of the Chief* in order to 1) provide timely an accurate view of what is happening in the theater of operations to the officers in charge and 2) send timely their orders to subordinates. In particular, they incorporate messaging capabilities and a map situation. In the following, the application of the method is exemplified, considering two consecutive iterations within the development process of a C2IS.

6.1. First Iteration

In the method's first step (Step 1 in Figure 1) three essential services related to information exchange are identified: *SendReport*, *RequestSupplies* and *TransmitOrder*. Figure 6 (left-hand side) shows the corresponding use cases (UCs) stereotyped `daService`[2].

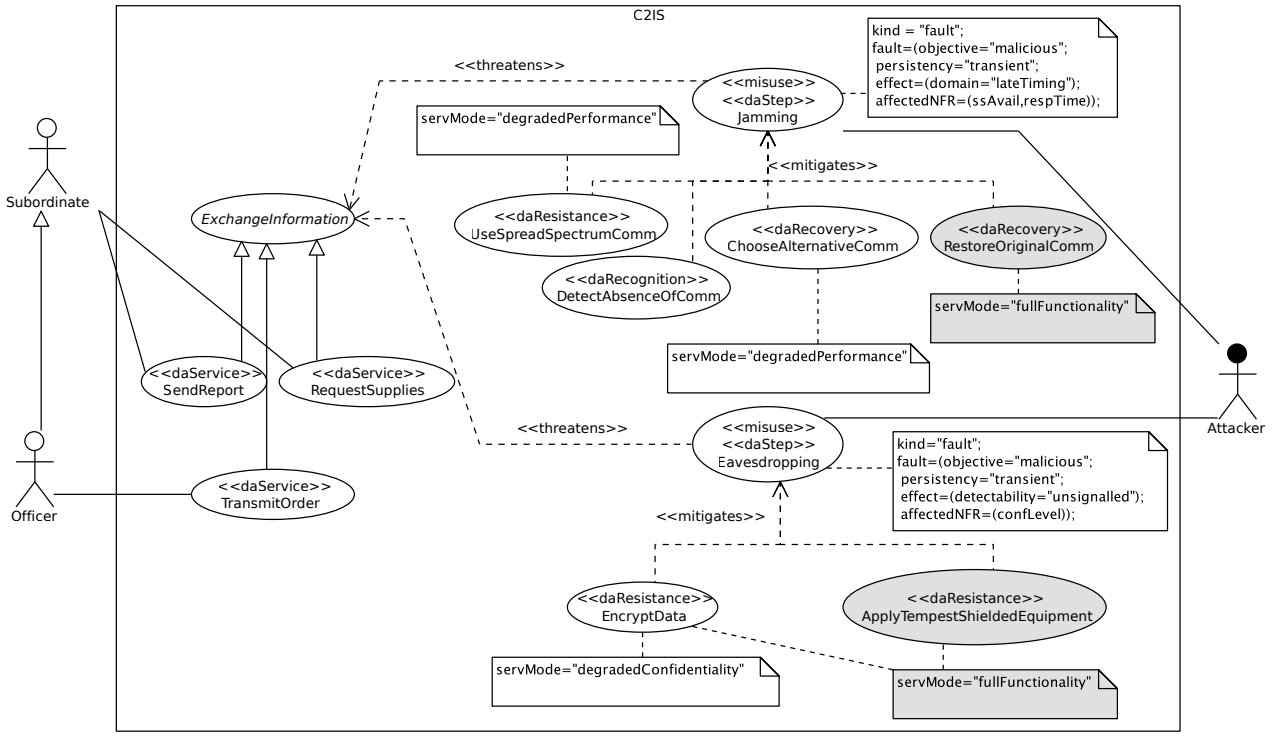

Fig. 6. Misuse case diagram (Iteration 1).

All these *ExchangeInformation* functionalities are characterized by QoS requirements, that are considered in the definition of the system service modes (see Table 5): *fullFunctionality* defines the required QoS under normal condition, i.e., assuming no threats affecting the essential services. The other two modes define the required QoS under degraded conditions, either considering *degradedPerformance* or *degradedConfidentiality*. The QoS metrics of interest are the steady state availability (*ssAvail*) and response time (*respTime*), while confidentiality is a qualitative indicator (*confLevel*) that enables to restrict the information exchange depending on the NATO clearance levels (i.e., top secret, secret, confidential, restricted). For example, a *high* confidence level indicates that the information can be exchanged at all clearance levels,

[2] To avoid cluttering, the figure shows only the essential services considered in this iteration.

daServiceMode fullFunctionality	daService SendReport	RequestSupplies	TransmitOrder
ssAvail	99%	99%	99%
respTime	(10,sec,max)	(10,sec,max)	(100,sec,max)
confLevel	high	high	high
daServiceMode degradedPerformance	daService SendReport	RequestSupplies	TransmitOrder
ssAvail	95%	95%	95%
respTime	(100,sec,max)	(100,sec,max)	(1000,sec,max)
confLevel	high	high	high
daServiceMode degradedConfidentiality	daService SendReport	RequestSupplies	TransmitOrder
ssAvail	99%	99%	99%
respTime	(10,sec,max)	(10,sec,max)	(100,sec,max)
confLevel	medium	medium	medium

Table 5. Specification of QoS for each service mode.

while a *medium* one limits the exchange of confidential and restricted information. Finally, metric (indicator) threshold values are assigned to the essential services: an exemplification is provided in Table 5.

In the method's second step (Step 2 in Figure 1) a vulnerability analysis is carried out first, to identify potential threats affecting essential services (Step 2.1: threats specification). Two types of attacks are considered specifically: sending radio signals to disrupt communication (*Jamming*) and accessing to the information exchanged between officers and subordinates (*Eavesdropping*). The attacks are represented by `misuse` cases (see Figure 6) and they are described in natural language using templates (see Table 6) and classified using the taxonomy in [28]. The result of such classification is specified in the diagram using tagged-values associated to the misuse cases, properly stereotyped `daStep`: e.g., a jamming attack is a *malicious transient* fault that causes delays in the information exchange (*lateTiming*), in particular it affects the QoS (*ssAvail*, *respTime*).

MUC Name	Jamming
Scope	C2IS
Level	Service goal
Main Misusers	Attacker
Success guarantee	The information is not delivered timely
Main scenario	The Attacker identifies the messaging system as a target: 1. The Attacker identifies the features of the communication link in use. 2. The Attacker sends an interference signal. 3. Communication is interrupted.

Table 6. Excerpt of detailed description of a jamming attack.

Once threats affecting essential services have been identified, survivability strategies to mitigate them are devised (Step 2.2: survivability strategies). Several strategies are required to mitigate a jamming attack (Figure 6, white UCs stereotyped `daResistance`, `daResistance` or `daRecognition`): use of spread spectrum communication (`daResistance` strategy that may not provide a 100% threat coverability), detection of absence of communication in case the resistance does not succeed (`daRecognition`) and consequent reconfiguration with an alternative communication mean (`daRecovery`). Both the above resistance and the recovery strategies should guarantee an acceptable degraded performance service mode (*servMode* tagged value in Figure 6). On the other hand, to mitigate an *Eavesdropping* attack only data encryption is required (`daResistance`) that may lead to a degraded confidentiality service mode in case of successful intrusion.

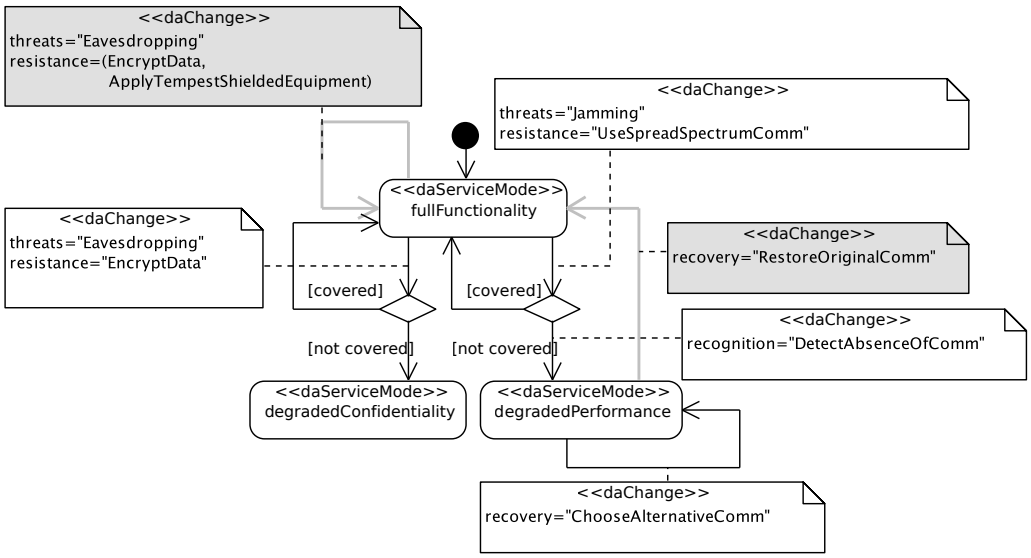

Fig. 7. SAM state-machine (Iteration 1)

In the next step of the method (Step 3: obtain a survivability assessment model) a SAM-SM is obtained from the misuse case diagram of Figure 6, by applying the patterns illustrated in Figure 2. The resulting SAM is shown in Figure 7 (white portion). In particular, the states (`daServiceMode`) represent the system service modes detailed in Table 5, the transitions (`daChange`) correspond to changes of service mode -due to threat occurrence and consequent survivability strategies execution. Observe that the tagged values associated to `daChange` transitions refer to (mis)use cases of Figure 6. Up to this point, the MASDES tool is primarily used to support the modelling.

Then, the SAM-SM is automatically transformed -by the MASDES tool- into a SAM-PN. The resulting PN model is illustrated in Figure 3(b) (see Section 4.4). The SAM-PN is used in the last step (Step 4: verify survivability properties) to verify, through model-checking techniques, whether the system requirements specification satisfies the survivability properties **PI1**, **PI2** and **PI3** described in Sub-section 5.3. In particular, MASDES detects that the recoverability property **PI1** is not satisfied. As a counterexample the tool shows that, once reached the *degradedConfidentiality* mode, it is not longer possible to recover to the *fullFunctionality* mode. This result suggests to require a stronger resistance strategy to *Eavesdropping* attacks, and the engineer decides to *ApplyTempestShieldedEquipment* (grey-colored use case in Figure 6) that, along with *EncryptData*, should guarantee a 100% of coverage. A new SAM-SM is then derived from the misuse case diagram and the survivability properties are checked again. The MASDES tool finds a new counterexample for property **PI1**, since it is not possible to leave from a *degradedPerformance* mode. Then, the engineer needs to introduce a new recovery strategy to restore the original communication, once a *Jamming* attack disappears[3] (*RestoreOriginalComm* stereotyped `daRecovery`, in Figure 6).

The SAM state machines may be appreciated in Figure 7: white portion corresponds to the initial SAM-SM; the last SAM-SM includes the grey transitions and removes the transition path from *fullFunctionality* to *degradedConfidentiality*. The tool determines that all the considered survivability properties are satisfied in the last SAM version, so the final misuse case specification can be used as input artefact for the design phase in this iteration.

[3] Observe that attack is a transient fault.

6.2. Second Iteration

Once finished the first iteration, the development process continues in a second iteration by identifying new essential services (Step 1 of the method). An essential service *CoordinateLandSeaAirOperations* and two other related to map management -*ConsultMap* and *UpdateMap*- are identified (Figure 8). All of them are stereotyped `daService` and their associated QoS requirements identified. Specifically, the considered functionalities are characterized by performance and confidentiality (QoS requirements shared with the functionalities analyzed in the previous iteration) and some of them by integrity (a new QoS requirement). The integrity metric of interest is a qualitative indicator (*integLevel*). A *high* integrity level would grant writing permissions to those processes that exchange critical, essential and routine information. The engineer considers the new metric in the scope of the service modes already defined and identifies a new degraded service mode *degradedIntegrity*. Initial threshold values are specified for each new essential service with respect to all service modes and metrics (see Table 7). Obviously, the values of the metrics associated to the essential services that were considered in the previous iteration remain unchanged.

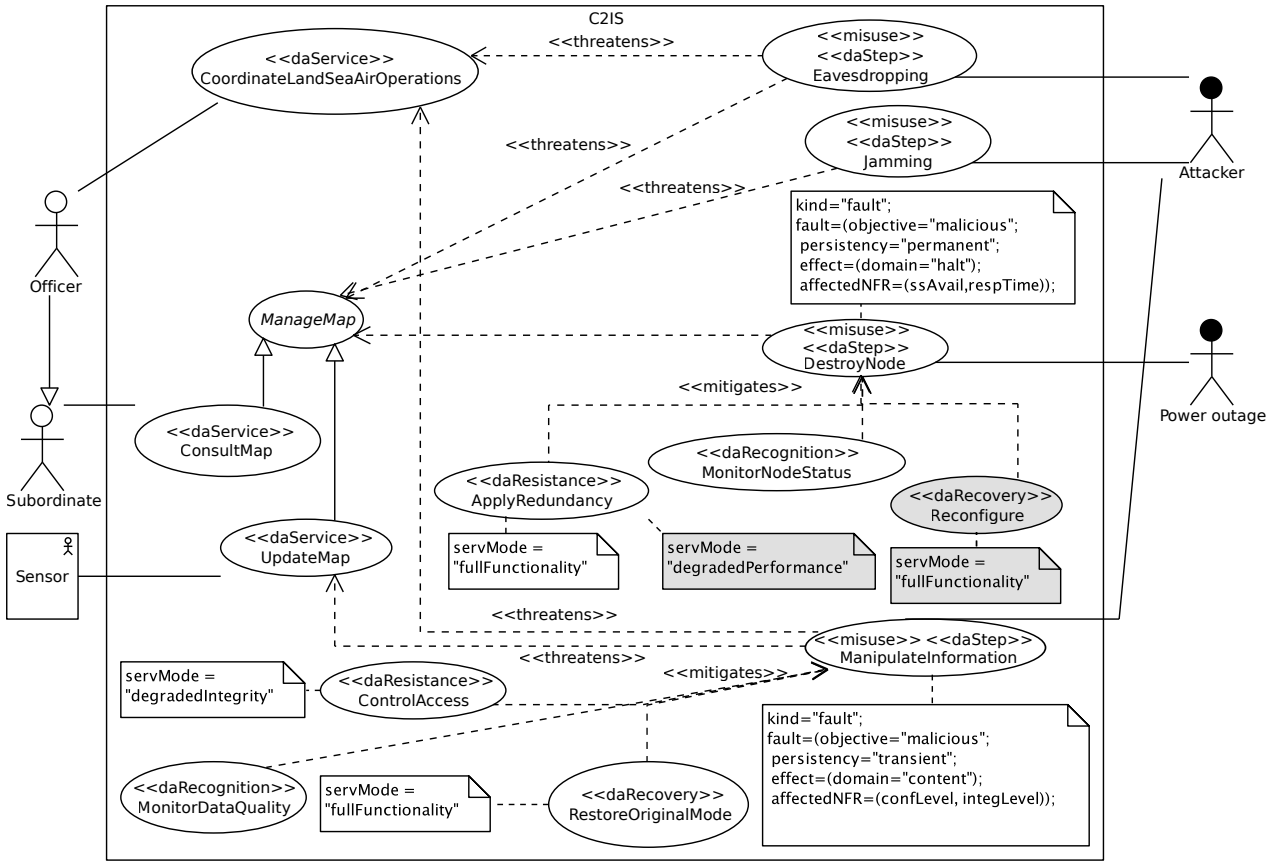

Fig. 8. Misuse case diagram (Iteration 2).

Next, threats to the new added essential services are identified (Step 2). The threats analyzed in the previous iteration (*Jamming* and *Eavesdropping*) affect also some of the essential services considered in the current iteration -the relation is reflected in Figure 8. Two new types of threats are considered: destroying a communication node (*DestroyNode*) and manipulation of the data sent by sensors or officers when updating the common situation map (*ManipulateInformation*). Each of them is specified in the diagram with the QoS metrics affected, e.g., the latter attack is a *malicious transient* fault that affects the content of the information exchanged (*content*), and consequently the QoS of the confidence and integrity metrics (*confLevel*, *integLevel*).

daServiceMode fullFunctionality	daService First iteration UCs	CoordinateLandSeaAirOperations	ConsultMap	UpdateMap
ssAvail	see Table 5	99%	99%	99%
respTime	see Table 5	(10,s,max)	(10,s,max)	(100,s,max)
confLevel	see Table 5	high	high	high
integLevel	-	high	-	high
daServiceMode degradedPerformance	daService First iteration UCs	CoordinateLandSeaAirOperations	ConsultMap	UpdateMap
ssAvail	see Table 5	95%	95%	95%
respTime	see Table 5	(100,s,max)	(100,s,max)	(1000,s,max)
confLevel	see Table 5	high	high	high
integLevel	-	high	-	high
daServiceMode degradedConfidentiality	daService First iteration UCs	CoordinateLandSeaAirOperations	ConsultMap	UpdateMap
ssAvail	see Table 5	99%	99%	99%
respTime	see Table 5	(10,s,max)	(10,s,max)	(100,s,max)
confLevel	see Table 5	medium	medium	medium
integLevel	-	high	-	high
daServiceMode degradedIntegrity	daService First iteration UCs	CoordinateLandSeaAirOperations	ConsultMap	UpdateMap
ssAvail	see Table 5	99%	99%	99%
respTime	see Table 5	(10,sec,max)	(10,sec,max)	(100,sec,max)
confLevel	see Table 5	low	low	low
integLevel	-	medium	-	medium

Table 7. Specification of QoS for each service mode (Iteration 2).

In this Step 2, survivability strategies for the new threats in the current iteration are also devised. First, the engineer specifies a daResistance strategy (*Apply redundancy*) and a daRecognition strategy (*MonitorNodeStatus*) to mitigate the threat of a communication node destruction (see Figure 8). These strategies should guarantee a 100% of coverage. Next, a *ControlAccess* is required as daResistance strategy to a *ManipulateInformation* attack. This strategy may lead to a *degradedIntegrity* service mode, in case the resistance fails. A daRecognition strategy (*MonitoringDataQuality*) and a daRecovery strategy (*RestoreOriginalMode*) should eventually bring the system back to full functionality mode.

In Step 3, MASDES tool is used to obtain a new SAM-SM, considering the new essential services only. This SAM-SM is used further by the tool to automatically build the corresponding SAM-PN.

In Step 4, the tool verifies the survivability properties on this last model. Specifically, MASDES tool detects that the recognition strategy *MonitorNodeStatus* does not satisfy the feasibility property (in Sub-section 5.3, the property **PT2** instantiated with the strategy *MonitorNodeStatus*). The engineer corrects the specification (Figure 8, grey part), so that the daResistance strategy *ApplyRedundancy* may lead to a *degradedPerformance* service mode and s/he adds a new daRecovery strategy (*Reconfigure*) that should eventually bring the system back to *fullFunctionality* mode.

7. Conclusion

This paper proposes a model-driven approach and a tool -*MASDES*- to assess the survivability requirements of critical systems. The method has been conceived to be easily reproduced in different software development processes (iterative and incremental, agile or prototype-based) by exploiting the use of (mis)use case technique and UML profiling for the survivability requirements specification. Concerning the verification of survivability requirements, the method exploits model-checking techniques that may suffer from the state space explosion. Even so the proposed method is scalable, since the size of the state space of the Petri Net model -which is automatically generated from the SAM-SM model- is directly proportional to the number of misuse cases. Considering the current available CASE tools and the model comprehensibility, such a number is reasonably expected to be low (i.e., less than 100) in a misuse case diagram. Moreover, each new iteration

gets a new SAM automatically, where only new essential services and new misuse cases are considered and, consequently, analyzed.

The MASDES tool has been developed to assist the proposed method and its main features are: 1) the full implementation of DAM and misuse case profiles in the Eclipse-Papyrus framework, that can be reused in other projects; 2) a modular software architecture that facilitates the addition of new Petri net model-checkers, currently two model-checkers are integrated; 3) a user-friendly GUI for selecting the properties to be verified; and, finally, 4) the extension of the PNML meta-model that enables the specification of Petri net logical properties.

The evaluation of the method and the tool -in Section 6- raised some limitations, that will be overcome in future extensions. In particular, regarding the analysis scope, we aim at extending it by introducing new properties, that also consider quantitative aspects (e.g., the essential services should be fully functional 99% of the time). With respect to the tool user-friendliness, the interpretation of the (counter-)examples produced by the model-checkers is currently a drawback. Indeed, one needs first to interpret the (counter-)example in the Petri net model and, second, to pinpoint the interpretation in the UML model. Automation of the analysis feedback implies to develop text-to-model transformations from the results produced by the model checkers to UML models. Although it is easy to integrate backward transformations in the tool (see grey plug-ins in Figure 4) due to its modularity, it is however challenging the research needed for getting the automated pinpoint.

Acknowledgements

Special thanks to the Lieutenant Colonel Félix Borque Pérez of the CASIOPEA centre at CENAD "San Gregorio" (Zaragoza, Spain) for his help in gathering the C2IS requirements and to María Berenguer that implemented the MASDES tool. This work was supported by the Spanish Ministry of Economy and Competitiveness [ref. TIN2011-24932 and TIN2013-46238-C4-1-R]; the Aragonese Government [ref. T27- DIStributed COmputation (DISCO) research group]; and the European Union Horizon 2020 research and innovation programme under grant agreement No. 644869 (DICE).

References

[1] Ellison RJ, Linger RC, Longstaff T, Mead NR. Survivable network system analysis: a case study. IEEE Software. 1999;16(4):70–77.

[2] Bernardi S, Dranca L, Merseguer J. Modelling and Verification of Survivability Requirements for Critical Systems. In: Canal C, Idani A, editors. In Proc. of SEFM2014 Collocated Workshops. vol. 8938 of LNCS. Springer Verlag; 2014. p. 1–15.

[3] Reisig W. Petri Nets. An Introduction. EATCS Monographs on Theoretical Computer Science. Springer Verlag; 1985.

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

[5] Sindre G, Opdahl AL. Eliciting security requirements with misuse cases. Requirements Engineering. 2005;10(1):34–44.

[6] Bernardi S, Merseguer J, Petriu DC. Model-driven Dependability Assessment of Software Systems. Springer; 2013.

[7] Jacobson I, Booch G, Rumbaugh J. The Unified Software Development Process. Addison Wesley; 1999.

[8] Eclipse [Online]. The Eclipse Foundation; [cited 30th January 2015]. Available from: `http://www.eclipse.org/`.

[9] Papyrus [Online]. The Eclipse Foundation; [cited 30th January 2015]. Available from: `http://www.eclipse.org/papyrus/`.

[10] ISO/IEC 15909-1: Systems and software engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation. International Electrotechnical Commission; 2004.

[11] ISO/IEC 15909-2: Systems and software engineering – High-level Petri nets – Part 2: Transfer format. International Electrotechnical Commission; 2011.

[12] Knight JC, Strunk EA. Achieving Critical System Survivability through Software Architectures. In: Architecting Dependable Systems II. LNCS 3069. Springer-Verlag; 2004. p. 51–78.

[13] Mustafiz S, Kienzle J, Berlizev A. 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. p. 19–28.

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

[15] Allenby K, Kelly K. Deriving Safety Requirements Using Scenarios. In: International Conference on Requirements Engineering. IEEE Computer Society; 2001. p. 228–235.

[16] Iwu F, Galloway A, McDermid J, Toyn J. Integrating safety and formal analyses using UML and PFS. Reliability Engineering and System Safety. 2007;92(2):156–170.

[17] Beato ME, Barrio-Solórzano M, Cuesta CE. UML Automatic Verification Tool (TABU). In: In Proceedings of SAVCBS'04. ACM; 2004. p. 106–109.

[18] Mazzini S, Latella D, Viva D. PRIDE: An Integrated Software Development Environment for Dependable Systems; 2004.

[19] Bozzano M, Cimatti A, Katoen JP, Nguyen VY, Noll T, Roveri M. Safety, Dependability and Performance Analysis of Extended AADL Models. Comput J. 2011;54(5):754–775.

[20] Johnson K, Calinescu R, Kikuchi S. An incremental verification framework for component-based software systems. In: Proceedings of the 16th ACM SIGSOFT Symposium on Component Based Software Engineering, CBSE'13; 2013. p. 33–42.

[21] Kwiatkowska M, Norman G, Parker D. PRISM 4.0: Verification of Probabilistic Real-time Systems. In: Gopalakrishnan G, Qadeer S, editors. Proc. 23rd International Conference on Computer Aided Verification (CAV'11). vol. 6806 of LNCS. Springer; 2011. p. 585–591.

[22] Bernardi S, Merseguer J, Petriu DC. A dependability profile within MARTE. Software and Systems Modeling. 2011;10(3):313–336.

[23] A UML profile for Modeling and Analysis of Real Time Embedded Systems, Beta 1; 2011. Adopted Spec., formal/2011-06-02.

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

[25] Alexander I. Misuse Cases: Use Cases with Hostile Intent. IEEE Software. 2003;20(1):58–66.

[26] Colom JM, Silva M, Teruel E. Properties. In: Girault C, Valle R, editors. System Engineering: a Petri Net based approach to modelling, verification and implementation. KRONOS; 1998. p. 41–63.

[27] Kruchten P. The Rational Unified Process: An Introduction. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.; 2003.

[28] Avizienis A, Laprie JC, Randell B, Landwehr C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing. 2004;01(1):11–33.

[29] Mehresh R, Upadhyaya S. Surviving advanced persistent threats in a distributed environment – Architecture and analysis. Information Systems Frontiers. 2015;p. 1–9.

[30] Dutheillet J, Ilié M, Poitrenaud D, Vernier I. State space based methods and model checking. In: Girault C, Valle R, editors. System Engineering: a Petri Net based approach to modelling, verification and implementation. KRONOS; 1998. p. 171–190.

[31] Clarke EM, Emerson EA, Sistla AP. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Lang Syst. 1986 Apr;8(2):244–263.

[32] Pnueli A. The temporal semantics of concurrent programs. Theoretical Computer Science. 1981;13(1):45–60.

[33] Varpaaniemi K, Heljanko K, Lilius J. PROD 3.2: An Advanced Tool for Efficient Reachability Analysis. In: In Proceedings of CAV'97. vol. 1254 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany; 1997. p. 472–475.

[34] MASDES software tool [Online]; [cited 30th January 2015]. Available from: `https://bitbucket.org/masdesgroup/survreq/wiki/Home`.

[35] Jouault F, Allilaire F, Bézivin J, Kurtev I. ATL: A model transformation tool. Sci Comput Program. 2008;72(1-2):31–39.

[36] Berthomieu B, Vernadat F. Time Petri Nets Analysis with TINA. In: In Proceedings of 3rd Int. Conf. on The Quantitative Evaluation of Systems (QEST 2006). IEEE Computer Society; 2006. p. 123–124.

[37] Diedrichsen L. Command & Control operational requirements and system implementation. Information & Security An international journal. 2000;5:23–40.