UNIVERSITÀ DEGLI STUDI DI FIRENZE
Dipartimento di Ingegneria dell'Informazione

Dottorato di Ricerca in
Ingegneria Informatica, Multimedialità e Telecomunicazioni
ING-INF/05

An Ontological Approach Supporting the Development of Safety-Critical Software

Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

in Informatics Engineering, Multimedia and Telecommunications

Irene Bicchierai

Ph.D. Coordinator
Prof. Luigi Chisci

Advisors
Prof. Enrico Vicario
Prof. Giacomo Bucci

XXVI Ciclo – 2011-2013

Ai miei genitori e a Leonardo

"...
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza."

Dante, Inferno, Canto XXVI, vv. 119-120

Acknowledgements

Innanzitutto vorrei ringraziare i miei Professori Enrico Vicario e Giacomo Bucci, per l'aiuto che mi hanno dato durante il lavoro di tesi, per avermi guidato dandomi spunti e suggerimenti durante questi tre anni, e soprattutto per la fiducia che hanno riposto in me.

Un grazie enorme a Laura. Per tutte le volte che ti ho chiesto aiuto e tu sei stata disponibile a darmelo, per tutte le volte che ci siamo ascoltate e capite. In questi anni mi hai insegnato tanto e spero di essere riuscita a mettere in pratica anche solo un po' dei tuoi insegnamenti.

Grazie a tutti i ragazzi che fanno parte, o hanno fatto parte, dell'STLab!!! Grazie a Lorenzo, Alessandro, Jacopo, Valeriano, Tommaso, Marco P., Marco M., Andrea, Fulvio, Simone. In particolare grazie a Carlo, con cui ho condiviso buona parte del lavoro di questa tesi. Grazie per le chiacchierate durante i pranzi in mensa, per i momenti piacevoli passati insieme, per i pranzi nel soppalco ...

E poi grazie alla mia famiglia, alle mie amiche, a Alessandra, alla mia cugina Giulia, per avermi sempre appoggiato e incoraggiato ad ottenere risultati sempre migliori.

Un grazie speciale ai miei genitori, Misia e Egidio, per come mi siete stati vicino in questi anni, per non avermi mai fatto mancare niente. Grazie per avermi assecondato in tutto ciò che mi piaceva e per aver accettato ogni mia scelta anche se talvolta non incontrava le vostre aspettative. Grazie perché se dovessi valere qualcosa come persona, il merito sarebbe soprattutto vostro.

E infine grazie a Leonardo, che, da fidanzato, in questi tre anni è diventato mio marito. Grazie per avermi sempre incoraggiato, sopportato e supportato, grazie per tutto l'amore che mi dai ogni giorno.

Abstract

In several application domains, the development of safety-critical software is subject to certification standards which prescribe to perform activities depending on information relative to different stages of development. Data needed in these activities reflects concepts that pertain to three different perspectives: i) structural elements of design and implementation; ii) functional requirements and quality attributes; iii) organization of the overall process. The integration of these concepts may considerably improve the trade-off between reward and effort spent in verification and quality-driven activities.

This dissertation proposes a systematic approach for the efficient management of concepts and data involved in the development process of safety critical systems, illustrating how the activities performed during the life cycle can be integrated in a common framework. This thesis addresses the exploitation of ontological modeling and semantic technologies so as to support cohesion across different stages of the development life cycle, attaching a machine-readable semantics to concepts belonging to structural, functional and process perspectives. The formalized conceptualization enables the implementation of a tool leveraging well established technologies aiding the accomplishment of crucial and effort-expensive activities.

Contents

List of Acronyms

AASTR	*APS Autonomous Star Tracker*
ADL	*Architectural Description Language*
APS	*Active Pixel Sensor*
ASD	*Astrium Space Deutschland*
BCCT	*Best Case Completion Time*
BDA	*Bi-Directional Analysis*
CENELEC	*European Committee for Electrotechnical Standardization*
CRC	*Class Responsibility Collaboration*
CRUD	*Create, Retrieve, Update, Delete*
CSCI	*Computer Software Configuration Item*
DAL	*Development Assurance Level*
DSML	*Domain Specific Modeling Language*
ECSS	*European Cooperation for Space Standardization*
ESA	*European Space Agency*
EU	*Electronic Unit*
FMEA	*Failure Mode and Effects Analysis*

FT	*Fault Tree*
FTA	*Fault Tree Analysis*
GSPN	*Generalized Stochastic Petri Net*
HCI	*Hardware Configuration Item*
IEC	*International Electrotechnical Commission*
IRC	*InfraRed Camera*
ISO	*International Organization for Standardization*
IT	*Image Tracking*
LOC	*Lines Of Code*
LQN	*Layered Queuing Network*
LS	*Laser Sensor*
MB	*Main Board*
MCS	*Minimal Cut Set*
MDD	*Model Driven Development*
MIL-STD	*Military Standard*
NMSPN	*Non-Markovian Stochastic Petri Net*
OS	*Optical Sensor*
OU	*Optical Unit*
OWL	*Ontology Web Language*
PA	*Process Algebra*
PN	*Petri Net*
POJO	*Plain Old Java Object*
pTPN	*preemptive Time Petri Net*
QN	*Queuing Network*

QoS	*Quality of Service*
RAMS	*Reliability, Availability, Maintainability and Safety*
RAMSES	*Reliability Availability Maintainability and Safety Engineering Semantics*
RDF	*Resource Description Framework*
RTAI	*Real Time Application Interface*
RTCA	*Radio Technical Commission for Aeronautics*
RTOS	*Real-Time Operating System*
SC	*System Control*
SD1	*System Requirements Analysis*
SD2	*System Design*
SD3	*SW-HW Requirements Analysis*
SD4-SW	*Preliminary Software Design*
SD5-SW	*Detailed Software Design*
SD6-SW	*SW Implementation*
SD7-SW	*SW Integration*
SD8	*System Integration*
SD9	*Transition To Utilization*
SDD	*Software Design Description*
SIL	*Safety Integrity Level*
SM	*Servo-Motor*
SMU	*System Monitoring Unit*
SPA	*Stochastic Process Algebra*
SPARQL	*SPARQL Protocol and RDF Query Language*

SRS	*Software Requirements Specification*
SSDD	*System/Subsystem Analysis and Design*
STD	*Software Testing Description*
SW	*software*
SW-FMEA	*Software Failure Mode and Effects Analysis*
SW-FTA	*Software Fault Tree Analysis*
SWRL	*Semantic Web Rule Language*
TE	*Top Event*
TPN	*Time Petri Net*
TS	*Temperature Sensor*
TVC	*TeleVision Camera*
UML	*Unified Modeling Language*
UML-MARTE	*UML profile for Modeling and Analysis of Real-Time and Embedded systems*
UML-SPT	*UML profile for Schedulability, Performance, and Time*
URI	*Universal Resource Identifier*
VP	*Video Processor*
W3C	*World Wide Web Consortium*
WCCT	*Worst Case Completion Time*
XML	*Extensible Markup Language*

Introduction

In the life cycle of safety-critical systems, development and documentation activities comprise a major step of the overall effort. While intended to support quality assessment along the entire life cycle, they become crucial in a more evident manner at the time of certification.

In industrial environments, the development life cycle of safety-critical SW is subject to specific certification standards, such as RTCA/DO-178B-C [94] for airborne SW, MIL-STD-498 [112] for military devices, CENELEC EN 50128 [34] for railways signalling, ECSS E-40 [45] for space, and ISO/IEC 62304 [64] for medical devices. These standards prescribe to perform activities and to produce documents collecting information scattered all along the development life cycle. Nature and form of practices and artifacts required by different standards share many common principles, and basically refer to the common framework of the V-Model lifecycle [30, 39].

The inherent complexity of prescribed activities is largely exacerbated by their dependency on information relative to different stages of development. In particular, these data are formalized in different documentation artifacts, often contributed by different parties or units, and pertaining to three different perspectives: the structural elements of design and implementation, the functional and *Reliability, Availability, Maintainability and Safety* (RAMS) requirements, and the organization of the overall process. The integration of these three perspectives may largely improve the trade-off between reward and

effort spent in verification and quality-oriented activities, and it may open the way to agile tailoring of the process model to the specific characteristics of different projects and organizations.

Among the activities prescribed by safety critical standards, *Failure Mode and Effects Analysis* (FMEA) and *Fault Tree Analysis* (FTA) aim at evaluating RAMS requirements, verifying the completeness of the countermeasures taken to mitigate the frequency of failures, by leveraging the analysis of the implemented functionalities and of the interactions among internal and external system components. More specifically, FMEA [73] is prescribed in industrial contexts to identify failure modes, their causes and effects, as well as determine actions reducing the impact of failure events. The analysis is carried out since the initial phases of the development process, when mitigating actions can be more easily taken. Unfortunately, FMEA-related information is usually acquired in natural language, implying that interpretation of the terms and concepts used across the analysis may differ from team to team; even the same team may give different interpretation when reusing an already performed analysis in a later occasion. Due to the lack of reusability, FMEA is often done from scratch. FMEA was first developed by the Department of Defence of USA and standardized in the MIL-STD-1629A [111], and it was then extended to various other industrial domains, some of which developed their own standards. FTA [119] is often carried out as the verification of FMEA. In fact, while FMEA is a bottom-up analysis, FTA is a top-down analysis that starts from a failure mode of the system and identifies conditions that would cause that failure mode. In other words, once a specific hazard is identified, FTA searches all possible combinations of the conditions that could force the system to reach the state of hazard.

Efficient and effective management of the volume and the heterogeneity of data involved in the development process have motivated research on methodologies for a systematic approach to execution, integration and correlation of all the activities performed during the product life cycle and of their relevant products (this whole concept is called *systematization*). The systematization of development practices may facilitate agile tailoring of the process model to

the specific characteristics of different projects and organizations, adapting to different constraints set by the applicable certification standards.

Ontological modelling and semantic technologies provide a relatively recent yet mature basis that may support this systematization aim. An ontology is defined as an explicit specification of a conceptualization [56], this means that it is used to formalize concepts involved in any domain of interest. In [51], three ontological models are proposed to characterize relations among components, functions, and quality attributes in complex embedded systems. The method and the system presented in [123] aim at facilitating reuse of knowledge as well as supporting complete and precise descriptions of processes and products. The semantic knowledge is hierarchically organized in form of taxonomies, containing typical recurring technical knowledge about systems, functions, failure modes, and actions. In [81], an ontology-based model-driven engineering process for compositional safety-analysis is introduced. The authors elaborate a domain ontology allowing the integration of a reasoner and inference rules to detect lack of model elements and inconsistent parts. In [41], the ontological formalization of FMEA concepts provides the ground for their explicit representation and for their unambiguous interpretation, while an ontology for the formalization of fault trees is proposed in [42]. In [70], a mapping from the concepts of an extended functional ontology to the concepts of a classic FMEA worksheet is introduced. The ontology presented in [16] enables to support the automation of activities and the management of information related to the FMEA process applied to SW (SW-FMEA), showing the effectiveness of the methodology in the context of a space project. Such ontological model is integrated with SW metrics in [15].

In this thesis, an ontological approach is proposed to support the development of safety-critical systems, also implementing a tool that enables its enactment in the framework of various certification standards. In particular, semantics of concepts and data involved in the development process is formalized, providing a common conceptual framework robust enough to enforce cohesion and consistency among information elements acquired along different phases of the development process and possibly contributed by dif-

ferent parties. The ontological model presented in this dissertation provides a self-consistent representation of concepts involved in three different perspectives: i) the structural perspective concerned with the structural decomposition; ii) the functional perspective concerned with functional requirements and quality attributes; iii) the process perspective concerned with the phases of the development and the documents produced. The framework permits to attach a machine-readable semantics to concepts collected in the life cycle and to tailor the model to the characteristics of different standards leveraging the extensibility and the manageability provided by the ontological architecture. The enactment of the ontological model in a practical application context is shown from two different points of view. On the one hand, the *UML profile for Modeling and Analysis of Real-Time and Embedded systems* (UML-MARTE) diagrams [84] and the *preemptive Time Petri Net* (pTPN) theory [26] are combined both to manage the documentation process prescribed by MIL-STD-498 [112] and to support design and verification activities of the development process. On the other hand, the ontological model is directly cast into an advanced SW architecture, built on top of well-established *Semantic Web* technologies. The formalized conceptualization enables effective application of reasoning tools aiding the accomplishment of crucial and effort-expensive activities.

The rest of the thesis is organized as follows:

- Chapter 1 recalls features of ontologies and Semantic Web technologies (Section 1.1), then describes the intensional part of the ontological model, made up by concepts comprised in the structural, functional and process perspectives (Section 1.2). Finally, SW-FMEA and SW-FTA are addressed, highlighting concepts of the model that are involved in these two techniques (Section 1.3).

- Chapter 2 illustrates an industrial tailoring of the V-Model SW life cycle, specifying the documents produced in each development activity (Section 2.1), then it describes the population of the extensional part of the ontology, illustrating the associations between concepts which provide the connections among the three perspectives (Section 2.2).

- Chapter 3 describes the application of the methodology to a project of development, introducing the context of application (Section 3.1) and showing how the documentation process is managed with UML-MARTE diagrams (Section 3.2) and how the design and verification activities are supported by pTPN theory (Section 3.3).
- Chapter 4 illustrates the tool supporting the methodology, through its architecture and its usage (Section 4.1), its functionalities, from basic (Section 4.2) to advanced (Section 4.3), and through the experience done within the real scenario of a scheduler of an electromechanical system for immunoenzymatic analyses (Section 4.4).

Chapter 1

Defining an ontology to systematize life cycle activities

The aim of this thesis is to provide a methodology to support the development process of safety critical systems, systematizing activities and practices typical of industrial contexts. This is carried out through an ontological model and a tool that leverages Semantic Web techniques.

This Chapter gives a hint of ontologies and ontological architectures (Section 1.1), describes the ontological model representing concepts involved in the development process (Section 1.2), and explains how these concepts support SW-FMEA and SW-FTA (Section 1.3).

1.1 Ontologies

An *ontology* is an explicit and formal specification of a shared conceptualization [56]. Ontologies are technologies for knowledge representation and constitute one of the main elements of the Semantic Web [12]. For a better understanding, ontologies should be evaluated in the context of the stack of

technologies and standards of the Semantic Web shown in Figure 1.1: *Ontology Web Language* (OWL) [77], the language standardized by *World Wide Web Consortium* (W3C) used to describe ontologies, depends on *Resource Description Framework* (RDF) [69] and RDF Schema [22] which, in turn, depend on *Extensible Markup Language* (XML) [21] and XML Schema [48]. Here only the main features needed to comprehend the proposed ontological architecture will be described.

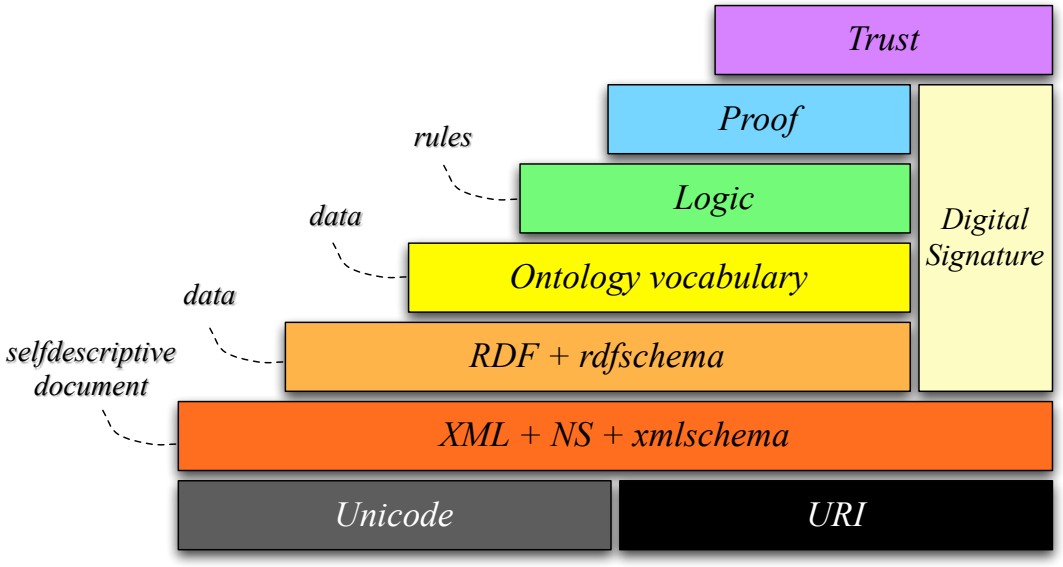

Figure 1.1. Stack of the Semantic Web technologies.

1.1.1 RDF ed RDF Schema

RDF [69] is the standard proposed by W3C for the encoding, the exchange, and the reuse of web metadata and is made up by two components: on the one hand the RDF Model and Syntax, which describes the structure of models and their representation in XML, on the other hand the RDF Schema, which describes the syntax needed to define schemas and vocabularies to represent metadata.

In a RDF model, every resource is identified by means of a *Universal Resource Identifier* (URI). The minimum unit which contains information is constituted by a *statement*, which is a triple of resources, shown in Figure 1.2, identified by a *subject*, a *predicate*, and an *object*. The object can be a datatype expression such as a string or a number as well as a resource.

Figure 1.2. The RDF triple.

A RDF model is constituted by some statements that can be connected by means of resources. The resource which represents the subject of a triple can be also, without any limitations, the object of another triple so as to construct an oriented graph. RDF models are extremely flexible because resources described in the models have not any special meaning. Only through RDF Schema a hierarchy of classes and a set of objects are defined through the specification of their meaning. RDF Schema is a vocabulary for describing properties and classes of RDF resources, together with semantics for generalization-hierarchies of such properties and classes.

1.1.2 Ontology Web Language

OWL [77] is the language proposed by W3C for the encoding of ontological models and is part of the stack of the technologies recommended for the Semantic Web. OWL is a vocabulary, which extends the vocabulary of RDF Schema, for the description of classes and properties of RDF Resources. Here only the main aspects of the language are described to give a compact and precise overview on its features and potentialities.

The fundamental elements of an ontological model are *classes*, which represent categories, sets or collections of elements (for example Doctor and Pa-

tient, but also Time and Space). Figure 1.3 shows how a class can be defined: through the specification of the name, using the notion of set, enumerating the elements which form the class, or applying a restriction on the value of a property. Classes can be related each other through inheritance, equivalence or disjointness.

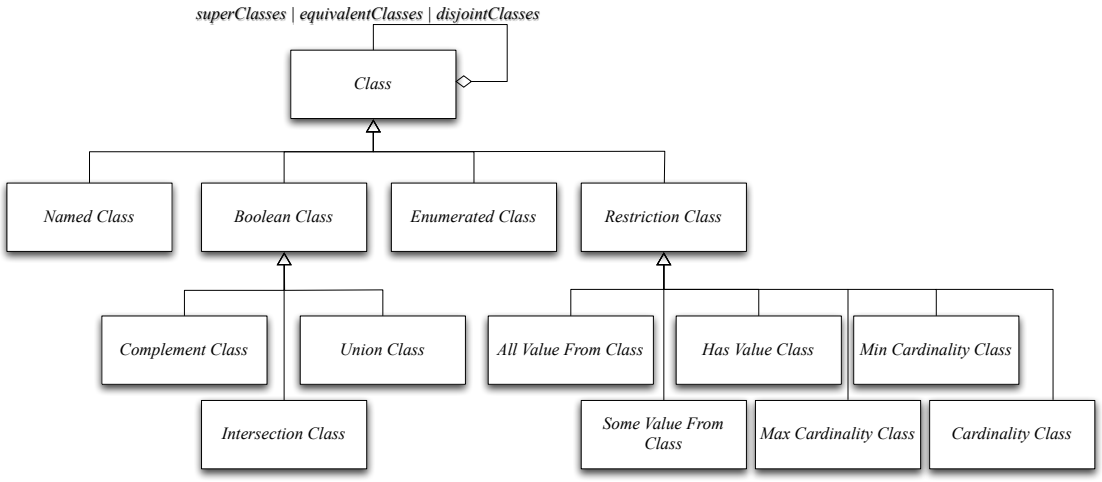

Figure 1.3. Simplified model of classes in OWL.

The other fundamental elements of ontological models are *properties*, which describe the internal structure of classes or the relations among classes. Figure 1.4 shows how properties and their features can be defined in OWL, for example they can be symmetric or transitive. Properties, such as classes, can be related each other through inheritance, equivalence or disjointness.

Classes and properties represent the *intensional* part of the ontology, while their instantiations represent the *extensional* part: *individuals* are realizations of concepts described by classes and *attributes* are realizations of properties.

The OWL language provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users:

- OWL Lite supports those users primarily needing a classification hierarchy and simple constraint features. For example, it provides support for thesauri and other taxonomies.

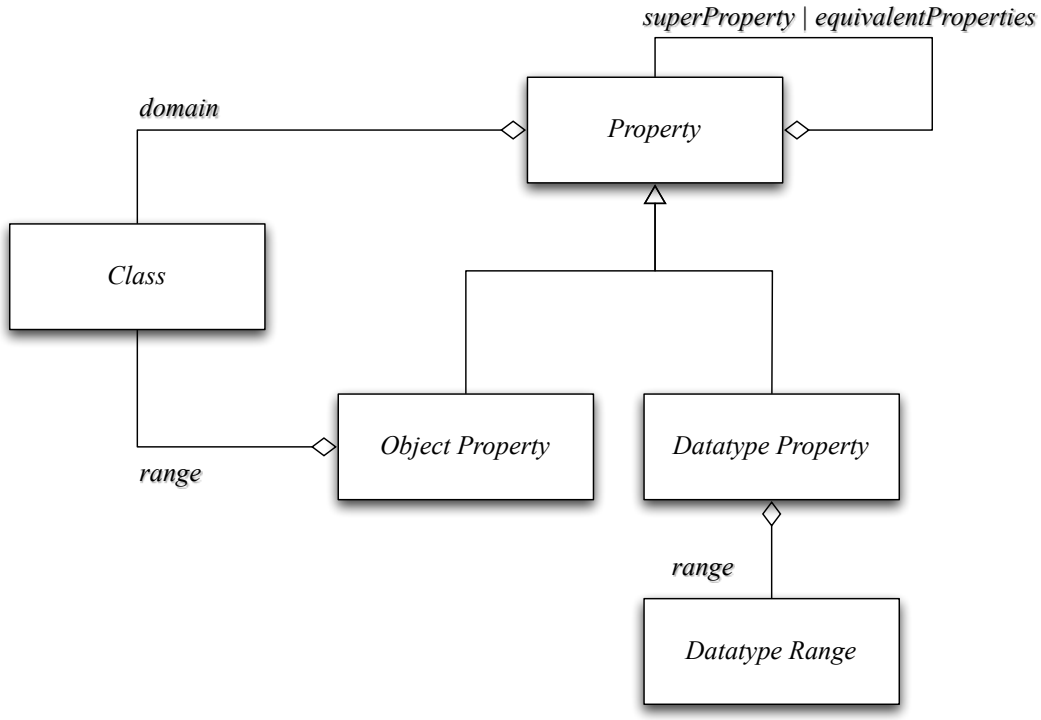

Figure 1.4. Simplified model of properties in OWL.

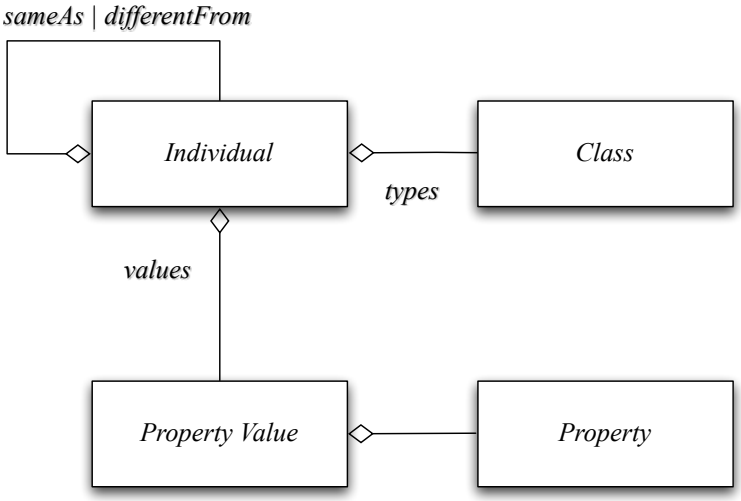

Figure 1.5. Simplified model of individuals in OWL.

- OWL DL supports those users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL DL includes all OWL language constructs with restrictions such as type separation (a class can not also be an individual or property, a property can not also be an individual or class). OWL DL is so named due to its correspondence with Description Logics, a field of research that has studied a particular decidable fragment of first order logic. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.

- OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded.

Recently, W3C has defined OWL 2 [120] which is an extension and revision of OWL, guaranteeing complete backwards compatibility with OWL 1. In fact, almost all the building blocks of OWL 2 were present in OWL 1, albeit possibly under different names.

1.1.3 SPARQL Protocol and RDF Query Language

SPARQL Protocol and RDF Query Language (SPARQL) [90] is a query language recommended for RDF by W3C. It can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or

RDF graphs.

Most forms of SPARQL query contain a set of triple patterns called a *basic graph pattern*. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable. A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph.

For example, Listing 1.1 shows a simple SPARQL query to find the title of a book from a graph of information. The query consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph. The basic graph pattern in this example consists of a single triple pattern with a single variable (?title) in the object position.

```
SELECT ?title
WHERE {
      <http://example.org/book/book1> <http://example.org/title> ?title .
      }
```

Listing 1.1. Example of a SPARQL query.

This query is supposed to be executed on the data graph shown in Listing 1.2.

```
<http://example.org/book/book1> <http://example.org/title> ‘‘First example’’
```

Listing 1.2. RDF model on which the SPARQL query is executed.

The result of the execution of the query is shown in Table 1.1.

title
“First Example”

Table 1.1. Result of the SPARQL query.

1.1.4 Semantic Web Rule Language

Semantic Web Rule Language (SWRL) [59] is a language proposed by W3C for writing inference rules and is a combination of OWL-DL with the Rule Markup Language [19]. It extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with an OWL knowledge base, allowing the application of deduction and inference rules on ontological models.

The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Both the antecedent (body) and consequent (head) consist of zero or more atoms. An empty antecedent is treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation; an empty consequent is treated as trivially false (i.e. not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple atoms are treated as a conjunction.

For example, using this syntax the rule shown in Listing 1.3 asserts that the combination of the *hasParent* and *hasBrother* properties implies the *hasUncle* property. In the abstract syntax the rule corresponds to the expression shown in Listing 1.4. It is worth noting that some rules that can be written through SWRL correspond to axioms that can be defined directly through OWL.

```
hasParent(?x1,?x2) and hasBrother(?x2,?x3) -> hasUncle(?x1,?x3)
```

Listing 1.3. Example of a SWRL rule.

1.1.5 Ontological architecture

Ontological technologies mainly originate with the intent to contribute to the realization of the Semantic Web [12]. This denotes an evolution of the current web, in which information is semantically defined so as to enable automated

```
Implies(
        Antecedent(hasParent(I-variable(x1) I-variable(x2))
                   hasBrother(I-variable(x2) I-variable(x3)))
        Consequent(hasUncle(I-variable(x1)I-variable(x3)))
        )
```

Listing 1.4. Example of a SWRL rule.

processing. Ontological technologies comprise a rich framework of paradigms, languages, and off the shelf components, which can serve beyond the specific intent of the Semantic Web, and may become an effective pattern for the organization of complex SW architectures. An *ontological architecture* [27] is a SW architecture with the capabilities of representation and classification of ontologies combined with the capability of elaboration of an objects-oriented language. In an ontological architecture there are three components:

- the *Domain Layer*, which realizes with an object model the application logic and the data processing functionalities;
- the *Data Layer* is responsible for data representation and conceptualization and is implemented through an ontological model;
- the *Mapping Layer* bridges the gap between the *Domain Layer* and the *Data Layer* solving the *impedance mismatch*, i.e. the conceptual distance between the object model and the ontological model, and enabling the application logic to operate both on the extensional and the intensional part of the ontology.

In an ontological architecture the Domain Layer can delegate to the Data Layer not only the representation of data but also the representation of concepts. In so doing, the domain logic is captured by the ontological model, enabling the generalization of the application logic so as to adapt it not only to the changes of instances, but also to the changes of concepts. This provides a method to dynamically modify the structure and the behavior of a system, as in the Reflection architectural pattern [103].

1.2 Ontological formalization of three perspectives

This Section illustrates how the formal characterization of concepts involved in the development process and the automatic manipulation of their data instances are supported by the ontological abstraction. These concepts belong to three different perspectives: i) the structural perspective concerned with the structural decomposition; ii) the functional perspective concerned with functional requirements and quality attributes; iii) the process perspective concerned with the phases of the development and the documents produced.

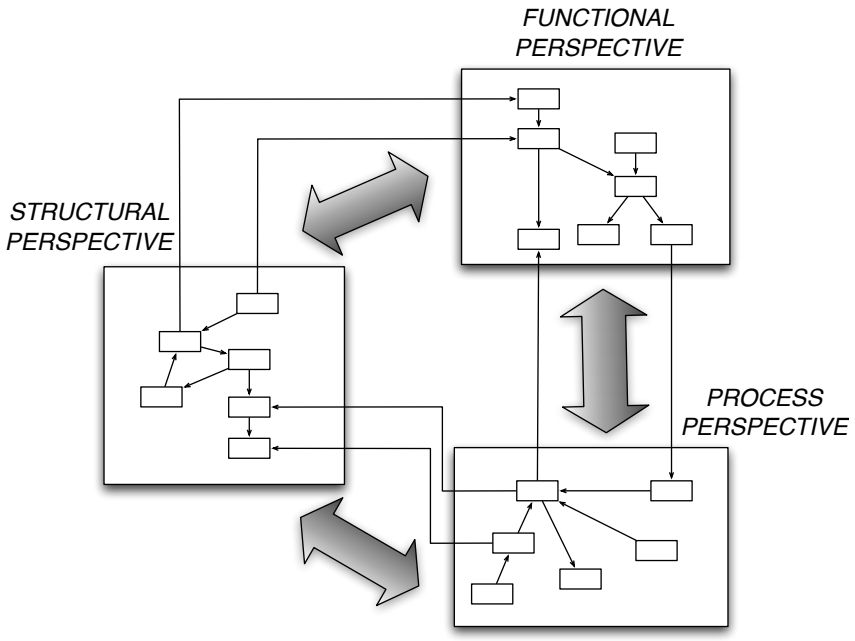

Figure 1.6. The whole ontological model representing all the concepts comprised in the three perspectives.

1.2.1 Structural perspective

The structural perspective, shown in Figure 1.7, comprises ontological concepts which model structural SW elements. A generic structural element is represented by the *Item* class. An item can be the entire *Computer Software*

Configuration Item (CSCI) (i.e. an aggregation of SW with an individual configuration), a *SW Component*, a *SW Module*, or a *Method*. Here items are hierarchically organized from the entire CSCI to the method (i.e. the smallest SW part with precise functionalities), however other types of structural elements can be added to the model as subclasses of the class *Item*. Furthermore items are linked through *is_part_of* or *is_composed_by* associations so as to allow the customization of the model according to the structural hierarchy adopted in the specific context.

Each *Method* is associated with *Code Metric*s which represent structural metrics of the code. Examples of code metrics are number of *Lines Of Code* (LOC), level of nesting, and cyclomatic complexity. Instances of code metrics are associated with a method through an instance of the *Code Metric Accountability* association class representing the value of the specific metric for that method. A structural item is associated with the *Fault*s that represent the structural defects of the item itself. A fault can be a *Basic Fault* or a logical combination of other faults (i.e. *AND*, *OR*, *NOT*).

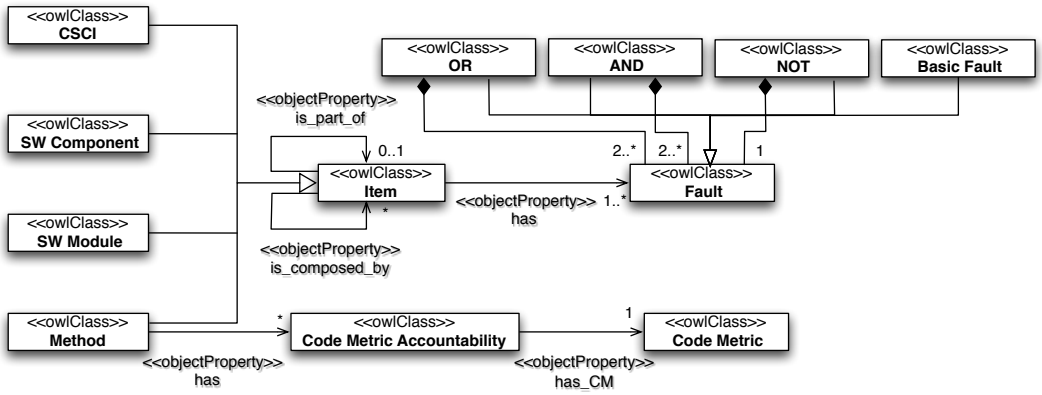

Figure 1.7. The ontological concepts belonging to the structural perspective.

1.2.2 Functional perspective

The functional perspective comprises ontological concepts concerned with functional and quality requirements. Figure 1.8 shows the involved ontological

concepts. A *Requirement* can be either a *Functional Requirement*, if it refers to the functionalities implemented by the CSCI or a *RAMS Requirement*, if it refers to RAMS attributes. A *Test*, associated with a requirement, verifies the correct implementation of it. A requirement is associated also with *Failure Event*s, that are the different ways in which the delivered service deviates from the correct implementation of the system function [2]. A failure event can be a *Testing Failure* or an *Operational Failure*, whether the failure is discovered during the testing phase or during the operational phase, respectively. A failure can be associated with the *Failure Effect*s produced on the system.

A requirement is associated with the *Assurance Level* which is defined depending on the risk associated with the implementation of the requirement itself. The assurance level must be satisfied in the development of SW elements implementing the requirement. Regulatory standards as [94, 34, 82] define the assurance levels classifying SW items according to risk associated with their implementation in relation to dependability attributes (e.g. availability, reliability, safety) relevant for the considered application [2, 101]. Standards prescribe the execution of activities and the production of artifacts along the development life cycle in order to achieve some specified levels of assurance: RTCA [94] defines *Development Assurance Level*s (DALs), which guide the activities performed along the development; CENELEC [34] associates with *Safety Integrity Level*s (SILs) the corresponding intervals of failure rates; NASA [82] specifies a *SW Risk Index* and a *SW Safety Effort* and the mapping between them. The allocation of the assurance level to SW can be performed through the use of algorithms and criteria as those discussed in [17].

1.2.3 Process perspective

The process perspective comprises the ontological concepts concerned with the development process such as activities and documents. Figure 1.9 shows the involved ontological concepts. Standards and regulation adopted in the specific context (e.g. CENELEC EN 50128 [34], ISO/IEC 62304 [64], RTCA/DO-178B [94]) are represented by the *Applicable Regulation* class. They prescribe

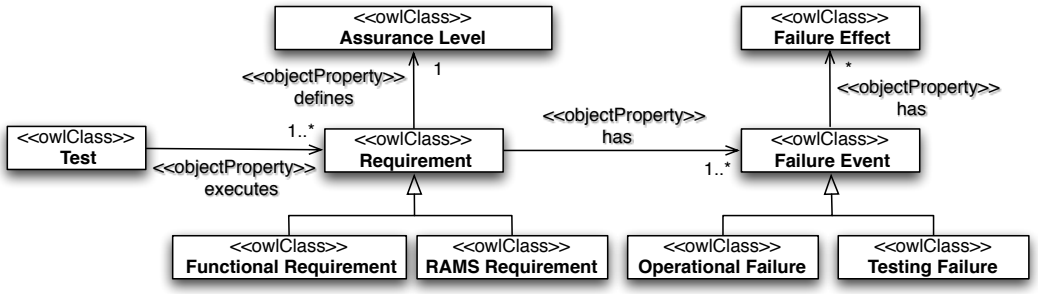

Figure 1.8. The ontological concepts belonging to the functional perspective.

to perform activities represented by *Design And Verification Activity* class, and guide the production of documents, represented by *Document* class, along the development life cycle. Examples of design and verification activities are testing, architectural verification with formal methods, Hazard Analysis.

In this thesis the MIL-STD-498 [112], which defines several types of documents with their content, is adopted. In Figure 1.9 four of the documents prescribed by MIL-STD-498 are modeled as subclasses of the *Document* class: the *System/Subsystem Analysis and Design* (SSDD), the *Software Requirements Specification* (SRS), the *Software Design Description* (SDD), and the *Software Testing Description* (STD). Nevertheless other documents can be modeled, adding one subclass for each document.

The SSDD document [115] describes the architectural design of a system or subsystem, identifying the CSCIs of the system and the requirements allocated to them, stating their purpose and presenting their mutual relationships. It is used as the basis for further system development.

In the SRS document [114], functional and non-functional requirements of a CSCI are described along with their failure modes and the effects of failures. Functional requirements are prescribed to be organized in groups: they can be divided depending on the operation mode they refer to. Following the common practice, an SRS can be divided in two parts, the one listing the functional requirements, the other summing up the quality attributes. The former part is modeled by a class labelled *CSCI Capabilities Reqs. Section*, the latter is

modeled by a class labelled *Other Reqs. Section*.

The SDD document [113] is divided in two parts: the former reports a high level description of the SW items and their functionalities, the latter provides a detailed description of the same items, including explanatory code fragments.

The STD document [116] describes test preparations, test cases, and test procedures to be used to perform qualification testing of a CSCI or a system or subsystem.

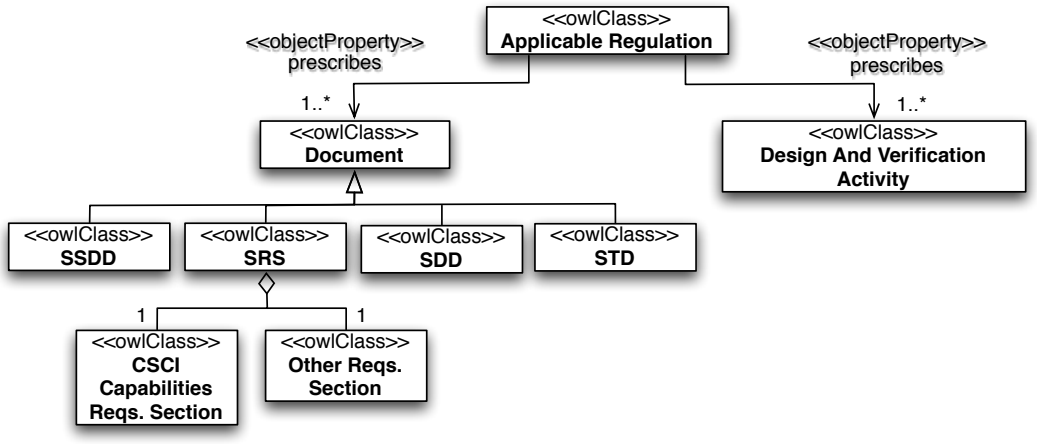

Figure 1.9. The ontological concepts belonging to the process perspective.

1.3 Supporting dependability techniques through ontologies

In this Section, two dependability techniques used in the development of safety critical systems are described, showing how the ontological model of Section 1.2 comprises concepts involved in these two techniques.

1.3.1 SW-FMEA and SW-FTA

FMEA is a widely used dependability and safety technique which aims at the systematic identification of the failure modes of a system, of the generating causes, of the consequences they might have, and of the countermeasures that could mitigate their effects or reduce their likelihood. FMEA was first developed by the Department of Defense of USA and standardized in the MIL-STD-1629A [111], and it was then extended to various other industrial domains, some of which developed their own standards [61, 106, 23]. Furthermore, FMEA has been treated in a large number of works in literature [108, 80].

Unfortunately, FMEA-related information is usually acquired in natural language, implying that interpretation of the terms and the concepts used across the analysis may differ from team to team; even the same team may give different interpretation when reusing an already performed analysis in a later occasion. Due to the lack of reusability, FMEA is often done from scratch and in large systems it is barely possible to avoid inconsistencies [41]. Furthermore, when used by itself, FMEA does not consider the combination of failure occurrences [49]. It is difficult to organize and master the large amount of information contained in the worksheets so there exist ad hoc tools aiding the FMEA process. Popular tools include: Relex [92], an integrated tool for reliability and safety evaluation of systems; XFMEA [98], which focuses on FMEA technique supporting data management and reporting; Byteworx FMEA [31], which handles FMEA process guiding the analysis with a checklist, pointing out the missing steps. In all these tools the main functionalities aim at providing a way to fill in FMEA worksheets and to navigate through the large amount of data. A less explored direction is how to deal with the volume and the heterogeneity of the data collected during FMEA at different phases of the life cycle.

To address these issues, several authors have proposed to resort to ontologies as a way to formalize the FMEA process, to manipulate concepts and to process data involved in the analysis. The methodology introduced in [72] is based on a knowledge engineering framework for integrating FMEA and

diagnosis models around a central Domain Ontology in support of multiple product-modeling environments. A reference model, unifying and formalizing FMEA concepts into an ontology expressed in F-Logic [68], has been proposed in [41].

While FMEA has been mainly thought for hardware systems, its use is also advocated for those systems where safety and dependability are strongly affected by SW, resulting in the so-called SW-FMEA. This is explicitly prescribed in several regulatory standards driving the development process in mature application contexts, such as space [47, 82] or railways signalling [34].

However, the application of FMEA to SW faces various major hurdles. Functional requirements allocated to SW are by far more complex than those assigned to hardware; identification of failure modes of SW components cannot rely on data-sheets or operational previous experience; SW faults often elude testing, remaining hidden until some complex triggering conditions activate them [2, 96]; they cannot be traced back to well identified causes, such as ageing, wearing, or stressing [87]. Complexities affecting SW-FMEA, in conjunction with those described for the classical FMEA, add further motivations to employ ontologies to deal with them, so as to improve the overall process.

*Fault Tree*s (FTs) are widely employed in the industrial practice [60, 62, 63, 91] as a means to represent the hierarchical relationships among causal factors that can yield an undesired outcome called *Top Event* (TE). A system can be modeled through a FT following a top-down approach, which identifies the events leading to the occurrence of the TE and expresses their relationships by combining them through boolean logic gates (i.e. AND, OR and KofN gates). This step is then repeated for each event until the so-called basic events are identified. Qualitative analysis of a FT provides the enumeration of the set of *Minimal Cut Set*s (MCSs), i.e. minimal combinations of leaf events that lead to the occurrence of the TE.

In the usual formulation, leaf events are associated with fixed, time-independent failure probabilities, calculated in a rather static manner on the basis of statistical information concerning the reliability of single components [119]. Quantitative evaluation of a FT supports reliability and safety analysis [57, 101]

through the derivation of the TE failure probability. This can be accomplished by following either an indirect approach, which derives the probability of the TE by combining the probabilities of all the MCSs, or a direct approach, which repeatedly combines nodes probabilities at each gate of the tree [95]. The probability of the TE depends only on the structure of the tree, with no reference to time. However, reliability of system components often evolves over time, due to such factors as: components aging; operation modes changing over time; maintenance and rejuvenation processes. In these cases, probability of the TE at different instants of time must be repeatedly recomputed in a kind of polling process, in order to take into account how the probability of component failures is conditioned by the actual operating conditions.

FTA, such as FMEA, is a technique originally thought for the analysis of hardware systems but it has been adapted to SW, resulting in the so-called SW-FTA. The subject of the analysis is the combination of causes that leads to a failure, represented in the root node of the fault tree, using boolean logic. The analysis can be conducted in several phases of the life cycle, even if the application to the source code level is extremely more complex and expensive. In fact, executing corrections and design changes during the first phases of the life cycle is more convenient than during the code implementation.

In [110], the application of SW-FTA during the phases of requirements analysis and system design is encouraged. In particular, the identification in this phases of most critical components, permits to apply for these components the SW-FTA also to the code level. During the phase of requirements analysis, the objectives of the application of SW-FTA are: a) identifying problems on the specification of requirements so as to modify weak requirements or add new ones; b) determining the requirements that have a strong impact on reliability or safety, so as to trace the SW components responsible for their implementation. During the SW design the objectives of the SW-FTA are: a) determining eventual flows or defects in the high level design; b) establishing SW components and modules implementing requirements identified in the previous phase, so as to make particular attention on their implementation. During the SW implementation, for every SW component identified in the previous phase FTs

are generated so as to determine the set of instructions that may lead to malfunctionings or hazards. In this way the SW-FTA is applied only to a limited, but significative, set of SW components, identifying the critical situations since the first phases of the development life cycle.

The *Bi-Directional Analysis* (BDA) [75, 50] is a technique which combines SW-FMEA and SW-FTA to provide a more complete analysis methodology to systematically identify the potential failure modes of the system, their effects and their propagation, and the events that trigger the failures. BDA starts with a forward analysis similar to SW-FMEA, proceeding from the failure modes to their effects so as to evaluate their consequences. Once failure modes with catastrophic consequences are determined, a backward search is performed through FTA, examining the probability of occurrence of each anomaly that has produced the failure mode. In this way the BDA determines the combination of vulnerabilities which can lead to failures.

The proposed methodology formalizes the semantics of both concepts and data involved in SW-FMEA and SW-FTA and it permits the automatic processing of data and the automatic generation of SW-FTs and SW-FMEA worksheets.

1.3.2 Ontological formalization of SW-FMEA and SW-FTA

The SW-FMEA process is naturally decomposed into phases. In [97], SW-FMEA is decomposed into two steps: i) a hazard analysis is initially performed to identify failure modes and their consequences; ii) countermeasures for mitigation or elimination of failures are then identified. In [53], SW-FMEA is decomposed into two phases: i) *System SW-FMEA*, performed early in the design process to minimize the impact of design recommendations resulting from the analysis; ii) *Detailed SW-FMEA*, performed later during the system design, with the aim of discovering safety requirements not being met and possible additional requirements. In [20], the process is decomposed into four activities: *Top level functional FMEA*, *Functional SW-FMEA*, *Interface SW-FMEA*, and *Detailed SW-FMEA*, corresponding to different levels of detail of the analysis.

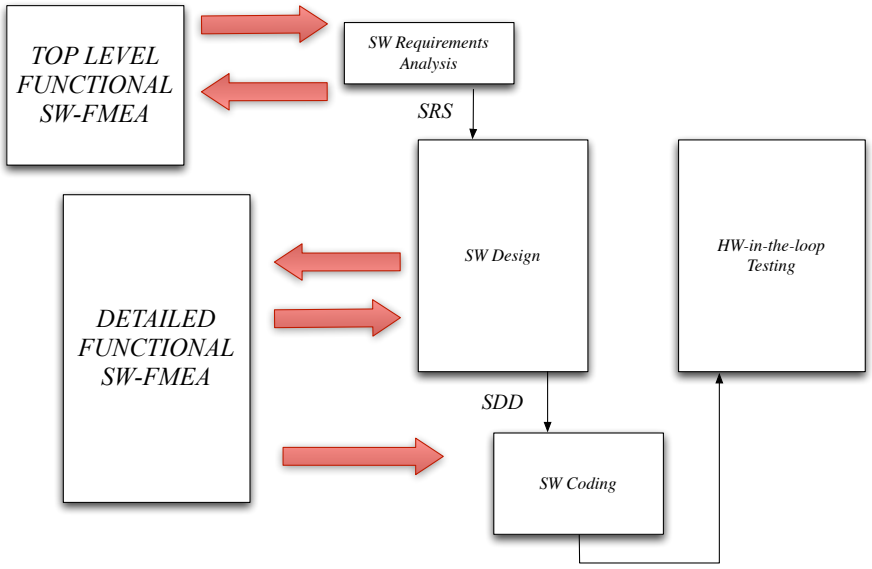

Figure 1.10. The stages of the SW-FMEA process mapped on the activities of a development life cycle.

Here, the SW-FMEA process is assumed to be decomposed into two phases spanning over the design side of the SW life cycle as shown in Figure 1.10: the *Top Level Functional SW-FMEA* is mapped on the *SW Requirements Analysis*; the *Detailed Functional SW-FMEA* is mapped on the *SW Design* and *SW Coding* activities. While the treatment of this thesis is focused on this concrete case, the proposed ontological model can be conveniently adapted to other methods and practices like those previously mentioned [97, 53, 20].

The first phase of the analysis, called *Top Level Functional SW-FMEA*, is carried out early in the SW life cycle, when the impact of changes to the original project is significantly less expensive. The analysis is performed during the allocation of technical requirements to the CSCIs in which the system is decomposed. The second phase of the analysis, called *Detailed Functional SW-FMEA*, is accomplished when the SW architecture is almost completed and each CSCI has been associated with a SW structure (see Figure 1.10).

The focus is on the definition of requirements associated with the *CSCI*s (in the Top Level Functional SW-FMEA) and the other structural items, such

as *SW Component*s, *SW Module*s, or *Method*s (in the Detailed Functional SW-FMEA). Requirements are first associated to CSCIs, then are decomposed and associated with structural items implementing them.

Each requirement is associated with one or more *Failure Event*s, which are the different ways in which the delivered service deviates from the correct implementation of the system function [2]. Failure events are associated with their *Failure Effect*s representing the consequences of the failure upon the system environment. Consequences are then classified through the severity of failure events. The number, the labeling and the characteristics of the severity levels are application-related and involve the dependability attributes for the considered application [46, 94, 35]. For example, according to the ECSS standard the severity level of a failure is classified in *minor* or *negligible*, *major*, *critical* and *catastrophic*. Each level is mapped to a numerical value ranging from 1 to 4 respectively. The causes of the failure events are represented by the *Fault* class. A fault can be a *Basic Fault* or a logical combination of other faults.

The criticality of each requirement accounts for the risk of failures and is defined as a function of the severity of the failure and the probability of the failure occurrence. The analysis is carried out in early development phases in an iterative manner. At each iteration the criticality of requirements is evaluated and, if necessary, further requirements are added as countermeasures against more severe failure events.

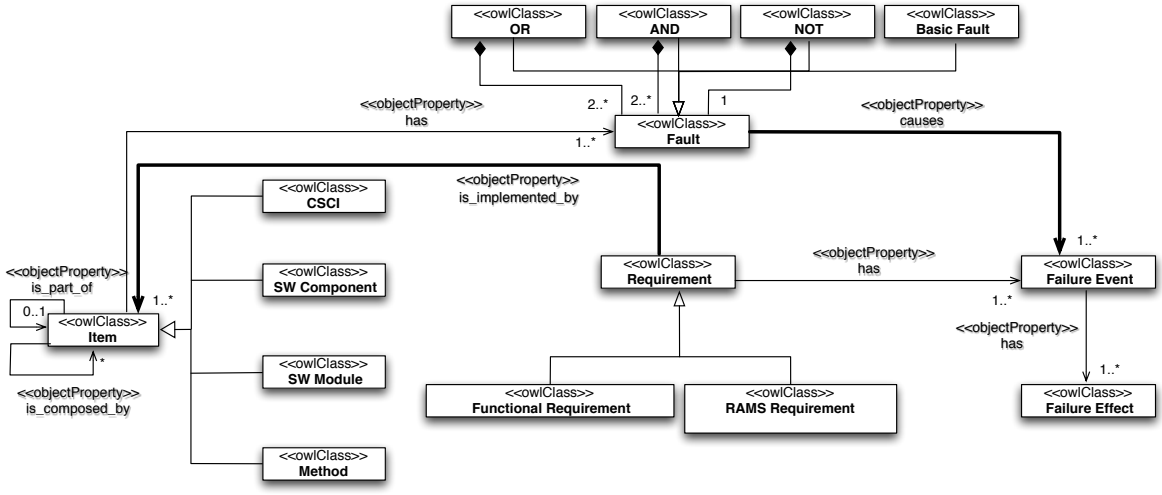

Figure 1.11. UML representation of the intensional part of the ontology modeling the concepts involved in SW-FMEA and SW-FTA.

Chapter 2

Instantiating the ontology to systematize life cycle activities

This Chapter describes an industrial tailoring of the V-Model SW life cycle [30], specifying the documents produced in each development activity (Section 2.1). Then the instantiation of the ontological model is described, showing how, in industrial practice, the extensional part of the model can be populated with actual data derived from artifacts and documents produced along the development life cycle (Section 2.2).

2.1 An industrial tailoring of the V-Model life cycle

Figure 3.1 shows the general structure of a V-Model SW life cycle [30] (inner scheme) and the specific industrial tailoring (outer scheme), emphasizing the artifacts of the documentation process prescribed by MIL-STD-498 [112] and possible iterative refinements along the development. The steps are briefly recalled to introduce the concepts that are significant for the proposed methodology.

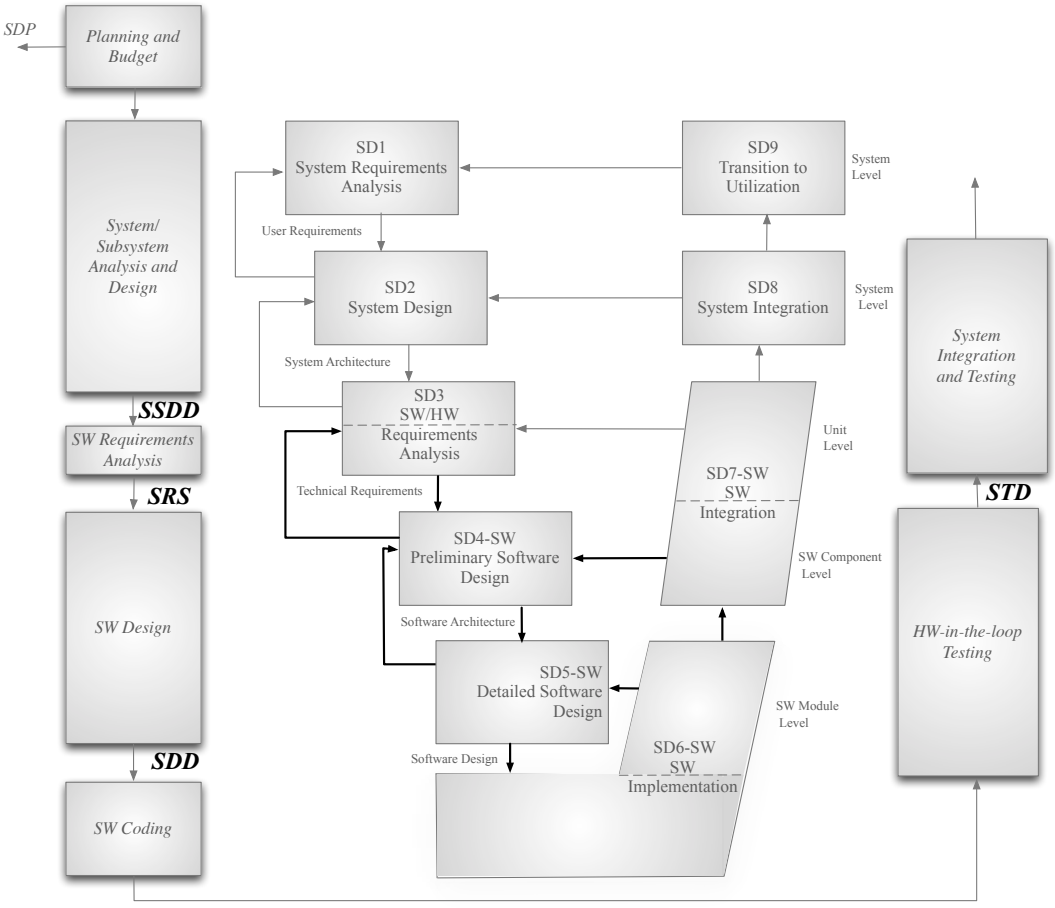

Figure 2.1. Scheme of the System Development (SD) submodel of the V-Model life cycle [30] tailored according to MIL-STD-498 [112]. The picture highlights MIL-STD-498 documents (in bold) and development iterations (bold arrows).

System Requirements Analysis (SD1), *System Design* (SD2), and the first part of *SW-HW Requirements Analysis* (SD3) are integrated in a single activity named *System/Subsystem Analysis and Design*. This develops on the outcomes of the *Planning and Budget* activity (out of the scope of the V-Model) and produces the SSDD document, specifying system decomposition into units made of CSCIs and *Hardware Configuration Item*s (HCIs). The second part of SD3 is mapped on *SW Requirements Analysis*, which lists all functional and non-functional SW requirements in the SRS document.

Preliminary Software Design (SD4-SW) and *Detailed Software Design* (SD5-SW) are integrated in *SW Design*, which produces the SDD document, specifying the *dynamic architecture* of each CSCI as a set of concurrent tasks (each of them is a SW component) with allocated resources and prescribed time requirements.

SW Implementation (SD6-SW) is covered by *SW Coding*, which implements the dynamic architecture of each CSCI and their functional behavior, and by the first part of *HW-in-the-loop Testing*, which addresses testing of low-level modules. *SW Integration* (SD7-SW) at the SW Component Level is mapped on the second part of *HW-in-the-loop Testing*, which verifies the integration of low-level modules within each CSCI and produces the STD which describes test preparations, test cases, and test procedures to be used to perform qualification testing of a CSCI.

SD7-SW at the Unit Level and *System Integration* (SD8) are aggregated in *System Integration and Testing*, which tests first the integration of CSCIs and HCIs within each unit and then the integration of all units within the system; *Transition To Utilization* (SD9) puts the completed system into operation at the intended application site. These activities are out of the scope of the industrial tailoring described here.

2.2 Connecting different perspectives

The proposed ontological formalization described in Section 1.2 provides a systematic ground for the integration of concepts concerned with the structural, functional and process perspectives. Concepts of the ontological model become concrete by associating them with actual data derived from artifacts and documents produced along the development life cycle. This stands for the population of the ontological model with instances of structural elements (e.g. CSCI, SW component), functional elements (e.g. functional requirement, failure event), process elements (e.g. document, applicable regulation), and relations among them. This Section illustrates the connections among the three perspectives showing how instances of associations between concepts of different perspectives can be obtained.

2.2.1 Tracing requirements

The association among structural and functional elements is a crucial point in the development and is also important with respect to maintainability, since it impacts on the ability of the system in isolating or correcting a defect as well as on satisfaction of new requirements. In addition, the identification of the SW items that implement a requirement is also relevant to verify that the implementation is compliant with design specification, in fact this activity is carried out after the SW Coding, where the code is already available. As a matter of fact, the regulatory standards expressly require that documents, such as SRS and SDD, contain the *traceability matrix*.

Traceability of requirements has been addressed in various works [44, 125, 124]. In [44] the identification of required computational units is performed through a technique that combines static and dynamic analysis, using concept analysis. The static part of the technique consists in the extraction of the static dependency graph, while the dynamic part of the technique traces the execution of some features, giving the system appropriate inputs. A classification of components is then obtained, analyzing the relevance of a component

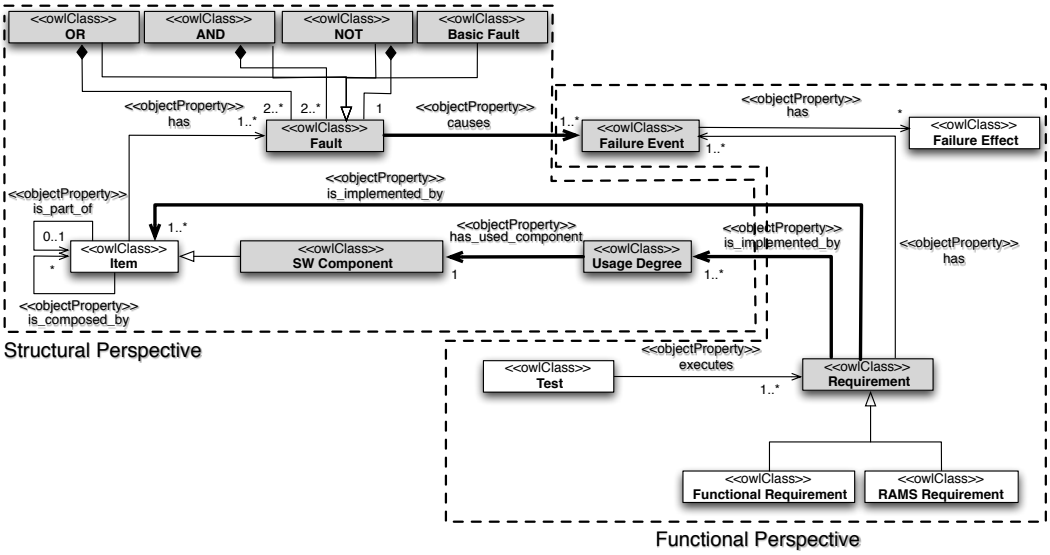

Figure 2.2. The connections among the structural and the functional perspectives, established through the associations between the SW items and the implemented requirements and between a failure event and the faults causing it.

with regard to a feature. The approach presented in [125] is based on the identification of code invoked during both the execution of the target feature and the execution of the other features. Subtraction of the second from the first gives the desired result. The quantitative evaluation of the relation between a component and a feature is addressed in [124] through the introduction of three metrics: the *concentration* of a feature in a component, the *dedication* of a component to a feature, and the *disparity* which measures the closeness between a feature and a component. The authors consider a component as a file composed by basic blocks (i.e. sequences of consecutive statements or expressions containing no transfer of control).

The objective of the analysis is the identification, through tests execution, of instances of the association between *Requirement*s and *Item*s shown (in bold) in Figure 2.2, which allows to connect the structural and the functional perspectives. Specifically, in order to show the approach, the association between a *Requirement* and *SW Component*s is considered. The association between a *Test* and *Requirement*s exercised in the test is also shown (in bold)

in Figure 2.2.

Formally, let r_i be a generic requirement and c_j a generic SW component, the relation $\mathcal{T} \subseteq 2^{FR} \times 2^{C}$ where FR is the set of requirements and C is the set of SW components, is looked for during the tests. The generic test $T \in \mathcal{T}$ is defined as $< T_r, T_c >$ where $T_r \in 2^{FR}$ and $T_c \in 2^{C}$. In doing so, $< T_r, T_c > \in \mathcal{T}$ means that a set of SW components T_c are related to a set of requirements T_r. Abusing of terms, a component c_i is called *necessary* for a requirement r_j if

$$\forall T \in \mathcal{T} : r_j \in T_r \Rightarrow c_i \in T_c,$$

a component c_i is called *potential* for a requirement r_j if

$$\exists T', T'' \in \mathcal{T} : r_j \in T'_r \wedge r_j \in T''_r \Rightarrow c_i \in T'_c \wedge c_i \notin T''_c.$$

As in [43], the usage degree (UG) of a SW component c_i in the implementation of a requirement r_j, accounts for how many methods M of c_i are executed in realizing r_j:

$$UG(c_i, r_j) = \frac{M(c_i, r_j)}{M(c_i)}.$$

To identify the relation $\mathcal{T}$, a set of tests is performed in order to exercise one or more requirements tracing the SW components implementing them. This has been done resorting to Aspect Oriented Programming [67, 107]. Instances of the association of Figure 2.2 between a requirement and SW components are obtained for each requirement extracting the components *necessary* for its implementation. Then these associations are imported in the ontology, which, as a result, provides the ground to perform verification of the compliance of SW implementation with its specification.

The connection between the functional and the structural perspectives is realized also with the association between a *Failure Event*, which represents a failure of the associated *Requirement*, and the *Fault*s that model the causes of the failure. The association between an *Item* and its *Fault*s cannot be completely automated. However, given a failure, thanks to the association of the requirement with SW items implementing it, the analyst is led to find those items that more likely contain the fault that causes the failure.

2.2.2 Following the documentation process

The specific instances of some concepts are reported in documents produced along the development life cycle and subject to MIL-STD-498 [112]. This enables the population of the extensional part of the ontological model while maintaining consistency with the industrial practice and improves cohesion among the activities of the life cycle and the documents, enabling their automatic production.

Figure 2.3 shows (in bold) the connections among the classes representing the documents and the classes corresponding to the concepts reported in the documents. Concepts belonging to the structural perspective addressing the SW structure are reported in design documents: CSCIs (i.e. *CSCI* class) are reported in the *SSDD*, while the other structural items (i.e. *SW Component*, *SW Module* and *Method* classes) are reported in the *SDD* where they are defined and associated with their functionalities. Concepts belonging to the functional perspective addressing requirements (i.e. *Requirement* class) are contained in the *SRS*. In particular, the former part of the SRS contains *Functional Requirement*s while the latter reports *RAMS Requirement*s. Concepts representing tests (i.e. *Test* class) are reported in the *STD*.

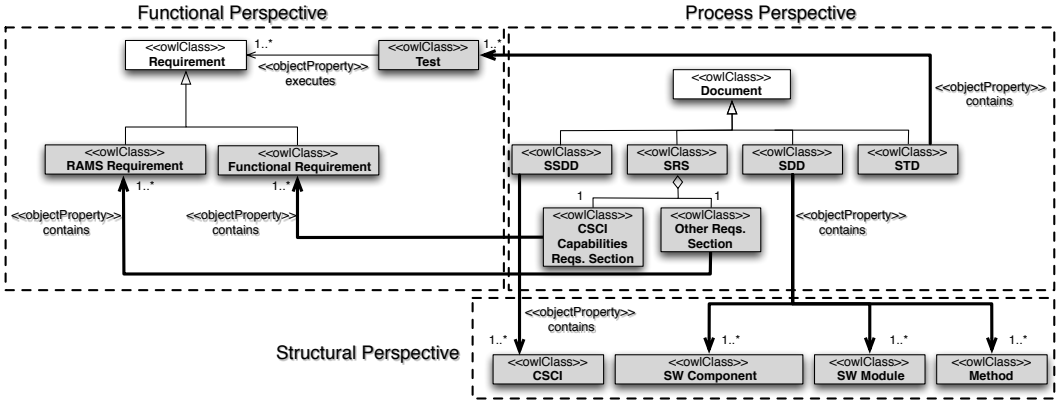

Figure 2.3. The connections among the three perspectives, established through the production of documents along the development life cycle.

2.2.3 Verifying activities of the development process

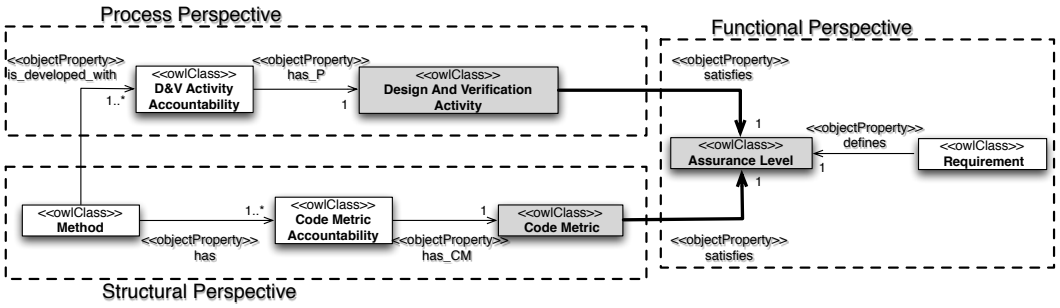

Figure 2.4. The connections among the structural, functional and process perspectives, established through the verification of the level of assurance.

The three perspectives are connected also through the verification of the level of assurance obtained in the development process, so as to guide the elimination of design and implementation flaws which could impair the system dependability. This is carried out after the implementation, by verifying whether the actual implementation of SW items satisfies the level of assurance associated with the requirements implemented. If this verification shows that the required level of assurance is not achieved, additional testing activities or code revisions of the items violating the expected level of rigor can be performed. A requirement defines a required level of assurance depending on the risk associated with the implementation of the requirement itself. The development of SW items implementing the requirement must satisfy the required level of assurance. For each level of assurance, standards prescribe to execute activities and to develop SW with specific values of code metrics. Therefore each level of assurance is associated with a set of required predicates about code metrics and design and verification activities [16, 15].

A set of predicates for a requirement r has the form

$$\mathcal{P}_r = \left\{ X_1 \lesseqgtr k_1, \ldots, X_n \lesseqgtr k_n, Y_1 = s_1, \ldots, Y_m = s_m \right\} \text{ with } n, m \in \mathbb{N}$$

where, referring to the fragment of the ontological model shown in Figure 2.4, X_i and Y_j, with $i = 1 \ldots n$ and $j = 1 \ldots m$, are instances of *Code Metric*

and *Design And Verification Activity* classes, respectively, while k_i and s_j are instances of *Code Metrics Accountability* and *D&V Activity Accountability* classes, respectively. If all the SW items contributing to the realization of r are implemented with values of X_i lower (greater) than k_i and Y_j equal to s_j, then r is considered rigorously implemented.

A concrete example of a set of predicates is

$$\mathcal{P}_r = \{CC < 5, TC = \text{“all edges”}\}$$

where CC stands for the McCabe's cyclomatic complexity and TC stands for the testing coverage, two metrics playing an important role in industrial contexts.

Predicates can be operatively verified by collecting values of code metrics and information about executed activities. Several tools supporting static analysis can be used to extract values of code metrics [117, 93], while data regarding activities can be extracted directly from the documentation. Once data relative to metrics are available, the validation process for a requirement implementation consists in checking whether each SW item implementing it satisfies the predicates. This will be shown in Section 4.3.

Chapter 3

Casting UML-MARTE and pTPNs in the ontology

This Chapter describes the application of the methodology in a experience of a development in a one-year-long project at the FinMeccanica site of Selex Galileo in Florence. The activities of the life cycle are supported by the combination of UML-MARTE diagrams [84] and pTPN theory [26] both to manage the documentation process prescribed by MIL-STD-498 [84] and to support development activities. The UML-MARTE diagrams allow to formalize the description of concepts contained in documents (Section 3.2). The pTPN theory allows to perform design and verification activities so as to obtain a determined level of assurance (Section 3.3).

3.1 Formal methods in an industrial SW process

Standards such as RTCA/DO-178B [94], MIL-STD-498 [112], CENELEC EN 50128 [34], ECSS E-40 [45], and ISO/IEC 62304 [64] explicitly recommend the introduction of formal methods as a means to improve the rigor of development

and the quality of SW, provided that the adoption of these techniques does not radically upset the consolidated practice. Hence, an increasing attention is focused on any measure that aims at smoothing the impact of formal methods so as to facilitate an effective integration within the development life cycle.

Formal methods can actually contribute to increase the quality of SW components by supporting multiple activities along the development life cycle. Formal modeling provides a well-defined semantics, which removes inconsistencies of natural language and permits definition of a non-ambiguous specification. This enables rigorous analysis through comprehensive exploration of system behaviors, supporting derivation of a proof of correctness of SW design. As a relevant point, early assessment of requirements allows early feedback at design stage, which may have an impact on the quality and the cost of the final product. Formal specification also supports *Model Driven Development* (MDD), including derivation of code that preserves model semantics, fast prototyping, incremental integration and testing of low-level modules. The formal description supports the testing stage as well, providing the basis for automation of the testing process and for generation of a test oracle.

UML-MARTE diagrams [84] and pTPN theory [26] are combined both to manage the documentation process prescribed by MIL-STD-498 [84] and to support development activities. In particular, UML-MARTE provides a semi-formal specification that is practical enough to meet the needs of an advanced industrial domain and sufficiently structured to allow mapping on pTPNs. This enables the integration of the two core processes yielding an effective ground for deployment of pTPN theory while attaining a smooth impact on the consolidated practice. Guidance for translation of UML-MARTE diagrams into equivalent pTPN models is provided, using timeline schemata as an intermediate artifact supplying a synthetic and compact representation of SW design.

Various efforts have been pursued to compile UML specifications [86, 85] into formal models used for performance prediction [3] and dependability analysis [10]. In many of these approaches, UML diagrams are translated into *Petri Net* (PN) models [79, 8, 9, 40, 7]. In [79], a compositional approach de-

rives a *Generalized Stochastic Petri Net* (GSPN) from a UML State Machine based on StateChart Diagrams, defining a formal semantics for a significant subset of State Machine elements. The approach is extended in [8] by applying the translation also to UML Sequence Diagrams, providing a more complete representation of system behavior. The method proposed in [9] combines State Machines and Activity Diagrams to derive a Stochastic Well-formed Net for evaluation of performance metrics, such as sojourn time and response time. In [40], performance of a SW architecture is evaluated through a two-phase methodology which first annotates a UML specification with tags and stereotypes of the *UML profile for Schedulability, Performance, and Time* (UML-SPT) [83], and then generates a corresponding *Non-Markovian Stochastic Petri Net* (NMSPN) model. In [7], a *Time Petri Net* (TPN) model is derived from a UML-based SW specification enriched with annotations of the UML-MARTE [84], which is specifically targeted to capture non-functional properties of real-time embedded systems. The resulting TPN model is used to assess the risk of timing failures in early stages of SW life cycle.

Several other approaches address translation of UML specifications into performance models based on *Queuing Network*s (QNs) and *Process Algebrae* (PA) [88, 37, 54, 76, 4, 105, 36, 89, 126, 55]. The approach proposed in [88] builds a *Layered Queuing Network* (LQN) from a UML description of system architecture made of Class/Object Diagrams and Sequence Diagrams, by converting each architectural pattern into a performance submodel. In [37], QN models are incrementally built from UML diagrams early available during SW development, providing fast feedback to the designer. The approach is extended in [54] to encompass mobility-based paradigms in the SW architecture of an application. In [76], a set of annotated Use Case, Activity, and Deployment Diagrams is translated into a discrete-event simulation model used to derive performance indexes. The methodology is improved in [4] using QN analysis to derive performance bounds. In the approach of [105], annotated UML specifications are exported and analyzed as QN models, using an XML-based interchange format which allows flexibility in design and analysis stages. SW performance analysis is also applied in [36] in the context of an industrial case

study from the telecommunication domain, translating Sequence Diagrams and StateChart Diagrams first into flow graphs and then into a specification based on Æmilia [11], an *Architectural Description Language* (ADL) defined upon a *Stochastic Process Algebra* (SPA). In [89], a metamodel named Core Scenario Model (CSM) is defined which supports derivation of various kinds of performance models from an UML diagram annotated with UML-SPT stereotypes. The approach is implemented in the PUMA (Performance by Unified Model Analysis) tool architecture [126], which provides a unified interface between SW design models and performance models. An intermediate metamodel is used also in [55] to derive performance models from UML diagrams. The transformation framework is based on a kernel language called KLAPER and helps in bridging the gap between design-oriented and analysis-oriented notations.

Integration of formal methods along the development process of real-time SW has been practiced in various MDD approaches and related tools [109, 1, 58, 18, 71, 32], supporting formalization of system requirements and design choices through *Domain Specific Modeling Language*s (DSMLs), and automated derivation of *concrete artifacts* such as real-time code, documentation, and tests [66, 102]. The model-based SW development process presented in [18] supports simulation and testing of complex embedded systems in automotive applications. To this end, an executable specification of the entire system is generated during early design phases and then iteratively refined throughout the design process. The Palladio model-driven approach [65] supports prediction of *Quality of Service* (QoS) properties of component-based SW architectures, providing a meta-model for specification of performance-relevant information [6] and a simulator for derivation of performance, reliability, maintainability, and cost metrics. It is implemented in a well-established tool which enables integration within a component-based development process [71]. In [32], an MDD framework is presented that integrates the core theory of pTPNs [26, 25] in a tailoring of the V-Model SW life cycle [30], enabling automated derivation of pTPN models from a semi-formal specification, automated compilation of models into real-time code running on RTAI [38], and measurement-based Execution Time evaluation. As a characterizing trait, pT-

PNs encompass temporal parameters varying within an assigned interval and support representation of suspension in the advancement of clocks. This attains an expressivity that compares with StopWatch Automata [33], PNs with hyper-arcs [100], and Scheduling-TPNs [74], enabling convenient modeling of usual patterns of real-time concurrency [29].

3.2 Supporting the documentation process through UML-MARTE

This Section shows how UML-MARTE [84] and other notations are used to support the documentation process prescribed by MIL-STD-498 [112] and to formalize concepts involved in the activities of the life cycle. Each Subsection describes an activity of the life cycle of Figure 3.1 together with the documents produced, which are instances of the *Document* class of the ontological model shown in Figure 1.9.

3.2.1 System/Subsystem Analysis and Design and SSDD document

During System/Subsystem Analysis and Design, definition of User Requirements enables identification of system functionalities and their allocation to system units. This activity produces the SSDD document which contains the description of the system decomposition into units and their decomposition in CSCIs and HCIs.

In the industrial case study addressed here, an electro-optical system is developed as a part of the equipment of a military vehicle to guarantee battlefield advantage through the use of visual, infra-red and thermal imaging, long-range target acquisition and illumination, and precise aiming. Therefore, the system is decomposed into: an *Optical Unit* (OU) made of sensors, cameras, and servo-motors; an *Electronic Unit* (EU) responsible for sensor control

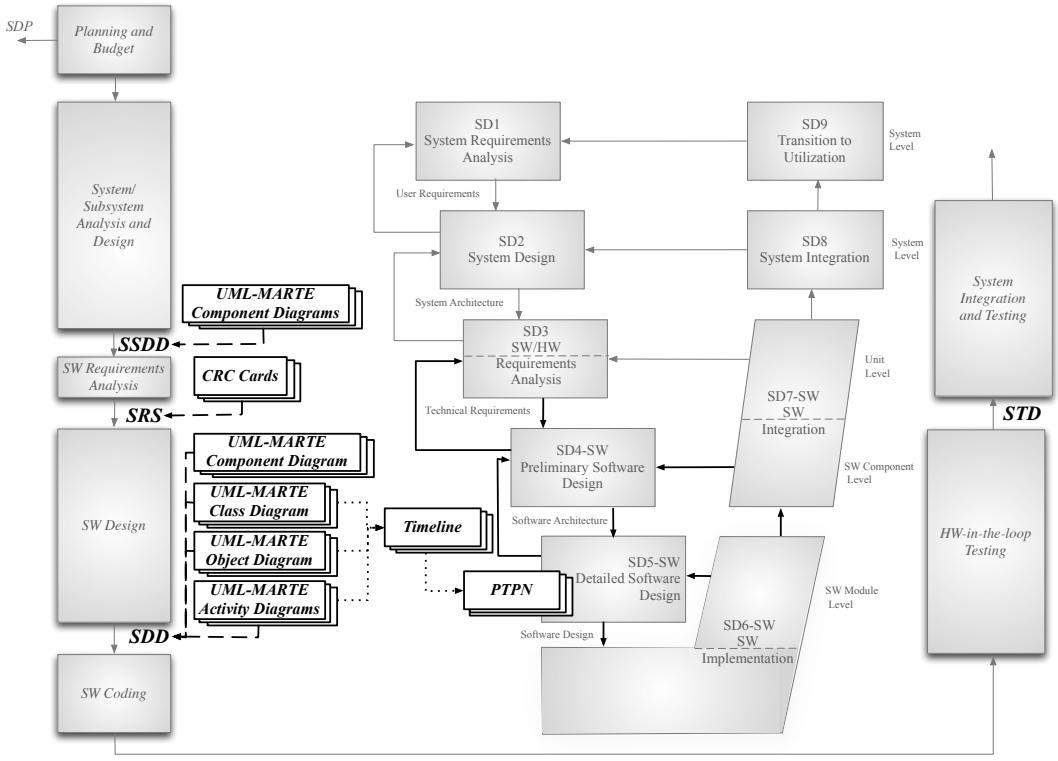

Figure 3.1. Scheme of the System Development (SD) submodel of the V-Model life cycle [30] tailored according to MIL-STD-498 [112]. The picture highlights the inclusion of artifacts (white boxes) within MIL-STD-498 documents (dashed arrows), translation of documentation artifacts into a formal specification (dotted arrows), and development iterations supported by the approach (bold arrows).

and image processing; and, a *System Monitoring Unit* (SMU) managing the entire system. EU plays the role of a bridge in the communication between SMU and OU, forwarding the commands periodically sent by SMU to OU and sending back the corresponding replies.

A UML-MARTE Component Diagram can effectively capture system decomposition, as exemplified in Figure 3.2: SMU and EU are modeled through the *ProcessingResource* stereotype (i.e., a resource allocated to the execution of schedulable resources); OU is represented through the *DeviceResource* stereotype (i.e., an external device that may be manipulated through specific services); and, the two communication channels connecting EU with SMU and OU are specified through the *CommunicationMedia* stereotype (i.e., a mean to transport information from one location to another).

Figure 3.2. System/Subsystem Analysis and Design: SSDD document. UML–MARTE Component Diagram of the system.

At this stage, functionalities of each system unit are allocated to CSCIs and HCIs. With regard to the industrial experience, the focus here is on the development of EU, which is sufficient to illustrate the approach and the complexities of the case study. According to this, the decomposition of OU into HCIs is illustrated only to make explicit the connections with HCIs of EU, and SMU is left out of the scope for this work. EU is responsible for controlling devices of OU, sending them commands elaborated by SMU and forwarding their replies to SMU. EU also processes images acquired by OU and sends obtained results to SMU. Therefore, EU functionalities are allocated to two CSCIs: *System Control* (SC), responsible for communication with OU and SMU, and *Image Tracking* (IT), responsible for image processing. In turn, each CSCI is associated with a real-time task-set and allocated to an HCI. Specifically: SC is allocated to *Main Board* (MB), which embeds a PowerPC MPC 5200B processor [52] and runs the VxWorks 6.5 [122] *Real-*

Time Operating System (RTOS); IT is allocated to *Video Processor* (VP), which runs a proprietary commercial RTOS. EU also embeds a battery. OU is made of six HCIs: *Servo-Motor* (SM), *InfraRed Camera* (IRC), *TeleVision Camera* (TVC), *Laser Sensor* (LS), *Optical Sensor* (OS), and *Temperature Sensor* (TS).

Also the aspect of unit decomposition can be effectively represented through a UML-MARTE Component Diagram, as illustrated in Figure 3.3. A CSCI is represented by the *rtUnit* stereotype (i.e., a real-time application that owns one or more schedulable resources); a real-time task-set is modeled by the *SchedulableResource* stereotype (i.e., an active resource that performs actions using the processing capacity brought from a processing resource managed by a scheduler); an HCI is specified through the *HwCard* stereotype (i.e., a printed circuit board typically made by chips and electrical devices); and, a battery is represented by the *HwPowerSupply* stereotype (i.e., a hardware component supplying the hardware platform with power). Association of a CSCI with a real-time task-set is modeled by the *allocate* stereotype (i.e., an allocation relation between elements of the application model and those of the execution platform, represented by the *ApplicationAllocationEnd* and *ExecutionPlatformAllocationEnd* stereotypes, respectively, denoted by *ap_allocated* and *ep_allocated* for short, respectively). The *allocate* stereotype is also used to represent allocation of a real-time task-set to an HCI.

An UML-MARTE Component Diagram also permits to make explicit relationships between HCIs and CSCIs as exemplified in Figure 3.3. Specifically, MB and VP are connected through bus $B7$, specified by the *HwBus* stereotype (i.e., a particular wired channel with specific functional properties). MB is also connected to SMU through bus $B6$ and to HCIs of OU through buses $B1$, $B2$, $B3$, $B4$, and $B5$. Note that every bus is bidirectional and is represented by a provided and a required interface in the diagram of Figure 3.3.

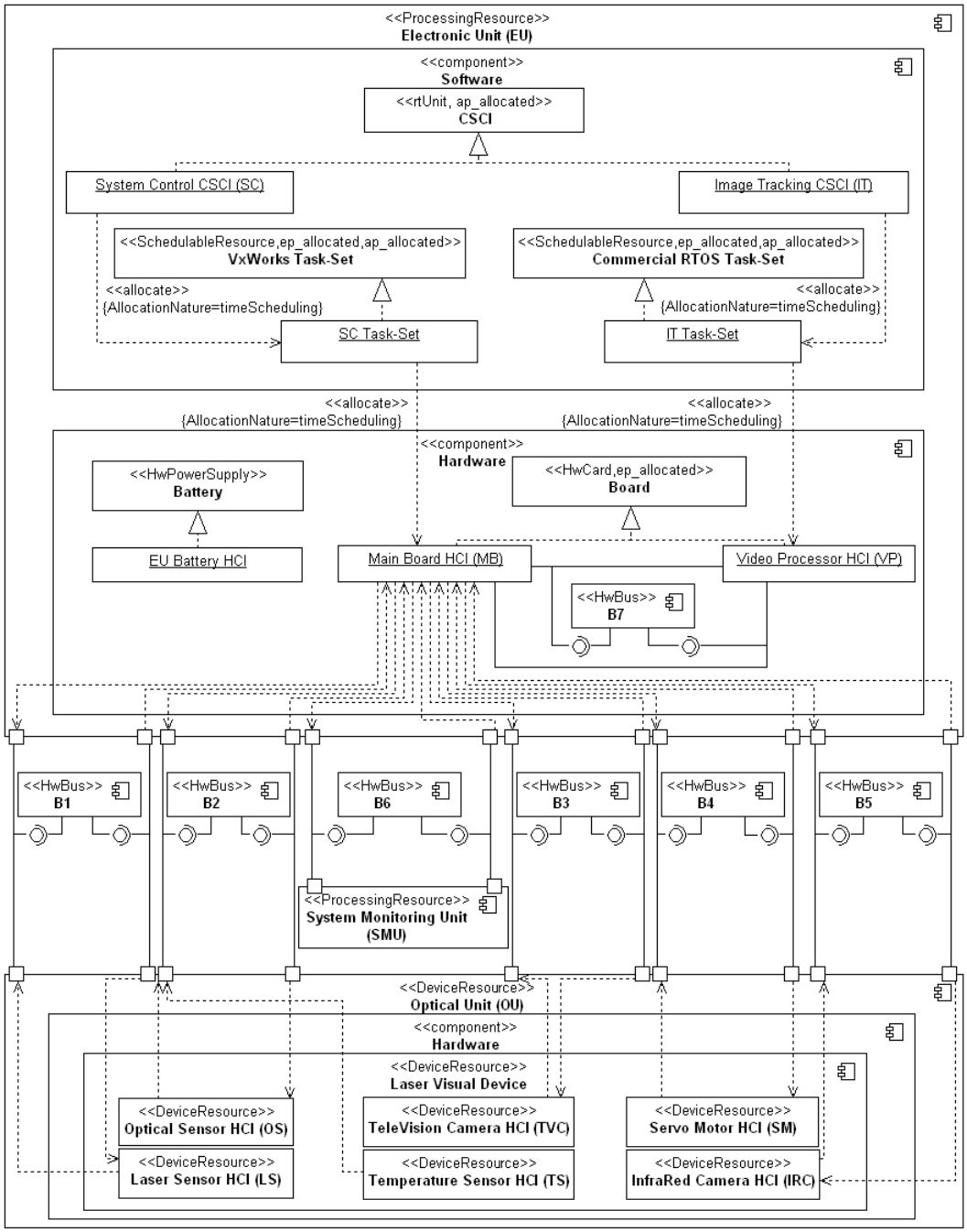

Figure 3.3. System/Subsystem Analysis and Design: SSDD document. UML–MARTE Component Diagram of system units.

3.2.2 SW Requirements Analysis and SRS document

SW Requirements Analysis and subsequent activities proceed separately for each CSCI up to the final integration. In the SRS document of a CSCI, a conventional structure similar to *Class Responsibility Collaboration* (CRC) cards [5] can be used to specify its functional behavior as a set of capabilities. Each capability reflects a CSCI functionality, is associated with one or more collaborating HCI/CSCI/Unit, and may be decomposed in sub-capabilities. Capabilities identified during this step are then allocated to structural items, enabling definition of their functional and non-functional requirements.

Table 3.1 specifies the capabilities of the SC, i.e., the CSCI of EU. *Init* initializes HW and SW; capabilities from *IT_Communication* up to *SM_Communication* manage buses connecting SC with: IT ($B7$), SMU ($B6$), and four HCIs of OU, i.e., LS ($B1$), IRC ($B5$), TVC ($B3$), and SM ($B4$); *LS-IRC_Power* and *TVC_Configuration* control LS, IRC, and TVC sensors; *SMU-OU_Commands* manages communication between SMU and OU and, since it requires the most computational effort, it is decomposed in sub-capabilities, shown in Table 3.2. Communication with the other two HCIs of OU (i.e., OS and TS) is performed through bus $B2$ which is directly managed by MB (i.e., the HCI which SC is allocated to).

3.2.3 SW Design and SDD document

In this context, the SDD document produced by SW Design specifies the *dynamic architecture* of a CSCI in the form of a set of concurrent tasks [29] following a pre-defined structure. Each task is represented in the ontology as an instance of SW component class of the model shown in Figure 1.7.

- A *task* may be either *recurrent* or *one-shot*. A recurrent task is an infinite sequence of identical activities called *jobs* activated with *periodic* or *sporadic* policy (i.e., with release time deterministic or bounded by a minimum value, respectively), with deadline less or equal to the minimum inter-release time. A one-shot task is a single job activated in

Capability	**Description**	**Collaboration**
Init	HW and SW initialization	-
IT_Communication	Communication with IT	IT
SMU_Communication	Communication with SMU	SMU
LS_Communication	Communication with LS	LS
IRC_Communication	Communication with IRC	IRC
TVC_Communication	Communication with TVC	TVC
SM_Communication	Communication with SM	SM
LS-IRC_Power	Switching on/off LS and IRC	LS and IRC
TVC_Configuration	Management of the TVC configuration	TVC
SMU-OU_Commands	Management of messages exchanged by SMU and OU	-

Table 3.1. SW Requirements Analysis: SRS document. CRC card of SC (the CSCI of EU).

reaction to an internal event (e.g., the release of a semaphore) or an external event (e.g., the arrival of a signal), with deadline less or equal to the minimum inter-occurrence time of the event.

- A job is a sequence of *chunks* running under *static priority preemptive scheduling*; each chunk requires one or more *resources* with a *priority level* (low priority numbers run first) and has an associated *entry-point* method implementing its functional behavior with a non-deterministic range of Execution Time.

- Chunks belonging to jobs of different tasks may share resources (e.g., memory space) and synchronize on semaphores under *priority ceiling emulation* [104], i.e., raising their priority to the highest priority of any chunk that ever uses that semaphore. Wait and signal semaphore operations are constrained to occur at the beginning and at the end of chunks, respectively.

Sub-Capability	**Description**	**Collaboration**
SMU_Commands	Management of SMU commands and OU replies	-
TVC_Commands	Management of the TVC configuration parameters	-
HCIs_Trasmission	Activation of data trasmission to IRC, TVC, and LS	-
LS-IRC_State	Management of the switched on/off state of IRC and LS	-
HCIs_Data	Processing of HCIs data	-
SM_LocationData	Processing of the SM location data elaborated by IT	-
OperationModes	Management of system Operation Modes	-

Table 3.2. SW Requirements Analysis: SRS document. Decomposition of *SMU-OU_Commands* in sub-capabilities.

3.2.3.1 Semi-formal specification through UML-MARTE

The model of a task-set can be conveniently documented through a UML-MARTE Class Diagram, as illustrated in Figure 3.4. Tasks are specified by the *SwSchedulableResource* stereotype (i.e., a resource that executes concurrently with other resources under the supervision of a scheduler according to a scheduling policy); chunks are modeled through the *EntryPoint* stereotype (i.e., a routine to be executed) and the association between a task and its chunks is modeled as a dependency; binary semaphores are represented by the *SwMutualExclusionResource* stereotype (i.e., a resource used to synchronize the access to shared variables).

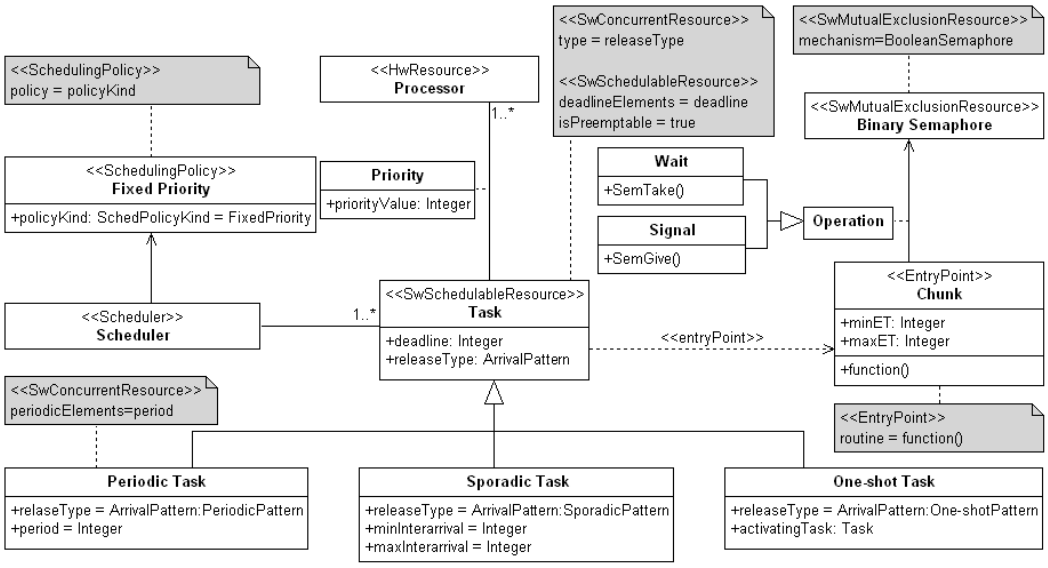

Figure 3.4. SW Design: SDD document. UML-MARTE Class Diagram of the task-set model.

The SW Design proceeds through three steps: *i)* identification of structural relations among tasks through a UML-MARTE Component Diagram; *ii)* definition of non-functional behavior of tasks through a UML-MARTE Object Diagram; and, *iii)* specification of functional behavior of tasks through UML-MARTE Activity Diagrams.

Specification of inter-task relations A UML-MARTE Component Diagram can be conveniently used to represent relations among tasks of a CSCI as well as relations between a task of a CSCI and the collaborating HCI/CSCI/Unit, as illustrated in Figure 3.5 with reference to the SC task-set of the industrial case study. Specifically, the six capabilities managing buses that connect SC with IT, SMU, and OU (in Table 3.2, Communication capabilities) are allocated to separate tasks named $Tsk2$, $Tsk3$, ..., and $Tsk7$; the remaining four capabilities (in Table 3.2, *Init*, *LS-IRC_Power*, *TVC_Configuration*, and *SMU-OU_Commands*) are assigned to a single task named $Tsk1$. According to this, the SC task-set is made of seven tasks and $Tsk1$ comprises its central element. $Tsk2$, $Tsk4$, $Tsk5$, $Tsk6$, and $Tsk7$ interface the associated HCI/CSCI with $Tsk1$ which, in turn, is interfaced to SMU through $Tsk3$. $Tsk1$ is responsible for processing SMU commands, producing data for the addressed HCIs/CSCIs, and writing them on shared memories. These data are then sent to HCIs/CSCIs by the associated tasks, which are also responsible for writing back the replies. Finally, $Tsk1$ forwards replies to SMU via $Tsk3$. Communications through buses are described by the *HwBus* stereotype, while shared memories are represented by the *SharedDataComResource* stereotype (i.e., a specific resource used to share the same area of memory among concurrent resources).

Specification of non-functional behavior In the definition of the dynamic architecture of a task-set, non-functional requirements are derived from contractual prescriptions, or obtained from previous artifacts, or autonomously chosen by the developer. Minimum inter-release times and deadlines are directly prescribed by User Requirements, while task periods are usually design choices. The number of chunks constituting a task reflects the number of sub-capabilities allocated to the task, and it may be refined during development iterations depending on the number of branches in the structure of the task. The Execution Time of a chunk can be first tentatively guessed through analogy with previous or similar realizations, and it is progressively refined during development iterations. Semaphore synchronizations necessary to ac-

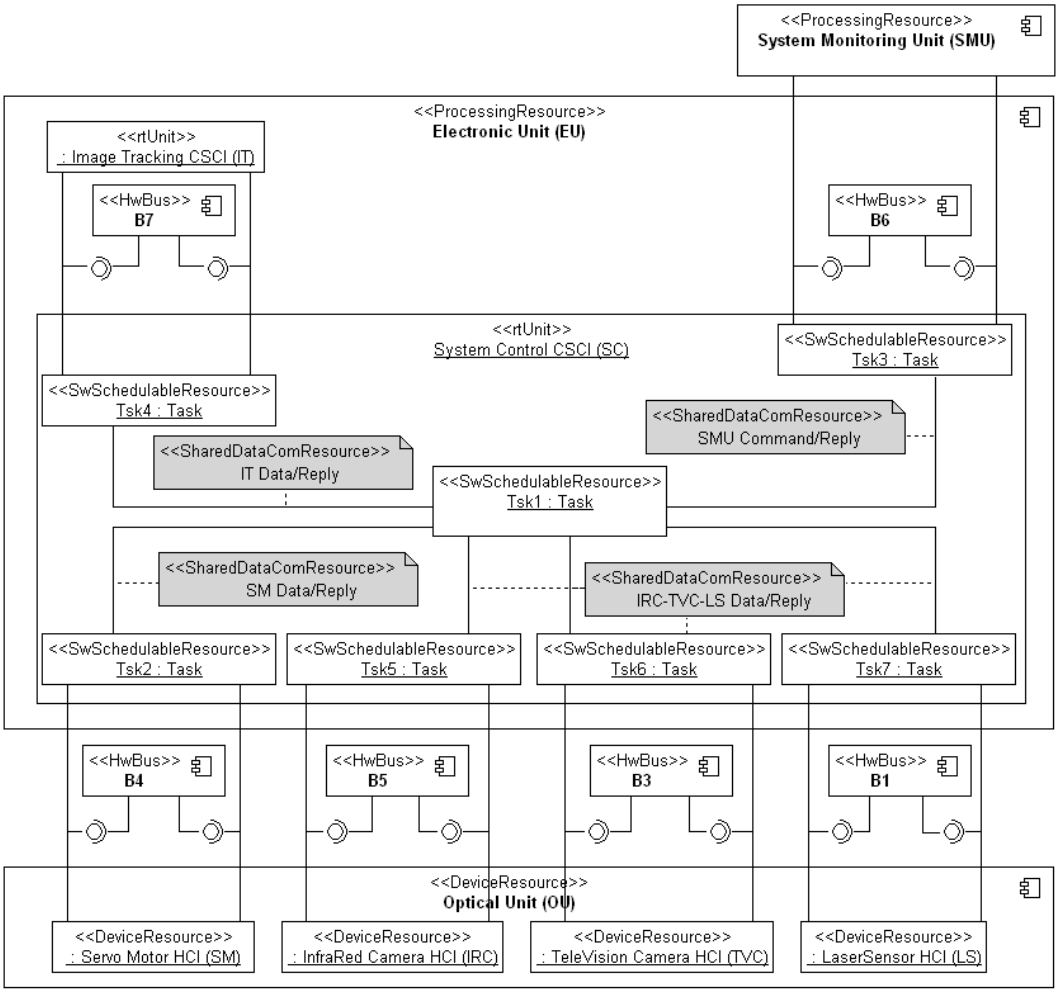

Figure 3.5. SW Design: SDD document. UML-MARTE component diagram of the SC task-set.

cess shared data directly come from tasks interactions.

A UML-MARTE Object Diagram can effectively capture the dynamic architecture of a task-set, enabling representation of non-functional properties, as exemplified in Figure 3.6 with reference to the SC task-set of the industrial case study. $Tsk1$ and $Tsk2$ are periodic tasks with period and deadline of 10 and 20 ms, respectively; $Tsk3$ and $Tsk4$ are sporadic tasks with minimum inter-release time and deadline of 20 and 40 ms, respectively; $Tsk5$, $Tsk6$, and $Tsk7$ are one-shot tasks with deadline of 10 ms. $Tsk1$ requires cpu with

priority level 1; the other tasks require *cpu* with priority level 2. Specifically, minimum inter-release times and deadlines of $Tsk3$ and $Tsk4$ directly come from User Requirements constraining timeliness of image processing and system management; $Tsk5$, $Tsk6$, and $Tsk7$ are modeled as one-shot tasks since communication with IRC, TVC, and LS is activated on demand by $Tsk1$, depending on the HCI/CSCI addressed by the current SMU command; $Tsk1$ and $Tsk2$ are modeled as periodic tasks to guarantee recurrent control on servomotors and SMU-OU messages; $Tsk2$ period and deadline are chosen equal to $Tsk3$ deadline so as to timely actuate SMU commands addressing servomotors; $Tsk1$ period and deadline are selected equal to half $Tsk3$ deadline as a result of the subsequent analysis.

A UML-MARTE Object Diagram also permits to specify the chunks of each task and their semaphore synchronizations. This is exemplified in Figure 3.6 with reference to the SC task-set of the industrial case study, avoiding representation of every chunk and semaphore to reduce the cluttering. For instance, $Tsk1$ is made of twenty-two chunks, which result from the four capabilities assigned to SC, the sub-capabilities of *SMU-OU_Commands*, and some branches introduced during refinement of entry-points; chunk $C11$ executes entry-point `f11` with an Execution Time constrained within $[0.005, 0.100]$ ms, and it is synchronized with chunk $C23$ on semaphore $sem1$ to access data that $Tsk1$ shares with $Tsk2$.

Specification of functional behavior The procedural aspects of a task-set can be conveniently specified using UML-MARTE Activity Diagrams according to the following methodology:

- each task is represented by a separate swimlane labeled with the task name;
- releases of periodic, sporadic, and one-shot task are modeled by input signals labeled with the period, the inter-release interval, and the activating event, respectively;

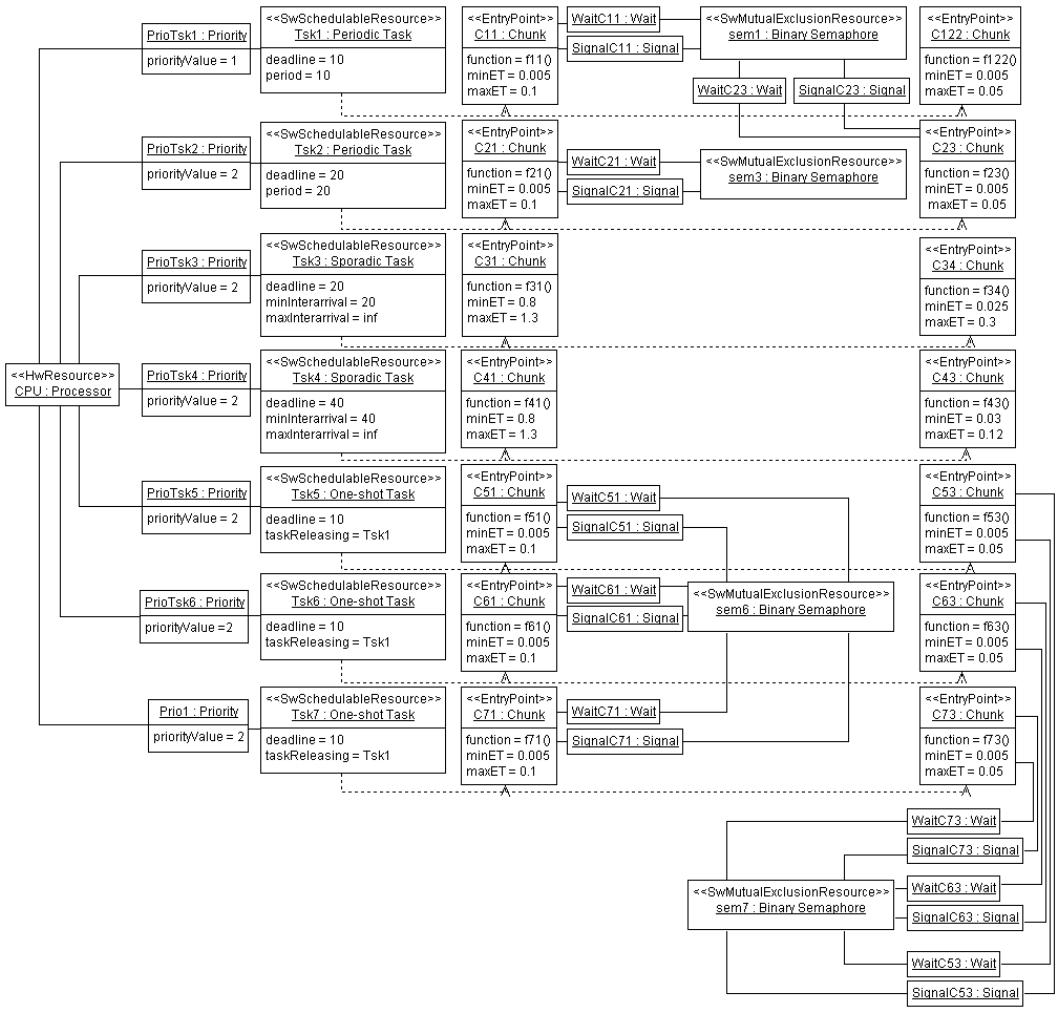

Figure 3.6. SW Design: SDD document. UML-MARTE Object Diagram of the SC task-set (times expressed in ms). For the sake of readability, only the first and the last chunk of each task are represented, e.g., $Tsk1$ is made of 22 chunks from $C11$ up to $C122$.

- chunk computations are specified by actions labeled with the chunk name;
- private data structures of a task are represented by objects lying within the task swimlane;
- data structures shared with other tasks are represented by objects lying on the border of the task swimlane;
- wait and signal semaphore operations are represented through input and output signals, respectively, labeled with the semaphore name.

Figure 3.7 illustrates the concept with reference to task $Tsk1$ of the industrial case study. The task is periodically activated every 10 ms; after activation, it performs the sequence of chunks $C11$, $C12$, $C13$, and $C14$, synchronizing on semaphores $sem1$, $sem2$, $sem3$, and $sem4$, respectively, to access shared memories. Afterwards, different paths are followed depending on OU and EU configuration parameters.

For reasons of space, UML-MARTE Activity Diagrams of $Tsk2$ through $Tsk7$, each composed of a task swimlane, are not shown here.

3.2.3.2 Semi-formal specification through timelines

In practical applications, UML-MARTE diagrams often tend to explode in complexity, as illustrated by Figures 3.6 and 3.7. To circumvent the problem, the methodology of development can conveniently integrate a domain specific notation based on the concept of timelines [29]. These provide a synthetic and intuitive description of the dynamic architecture, acting as an intermediate model that helps in bridging the gap between a semi-formal specification suitable for SW documentation and a formal specification supporting correctness verification through analysis. In this perspective, the proposed approach exposes similarities with [55, 40, 89], where intermediate artifacts are used as an interface between design-oriented and analysis-oriented models. In the formalism of timelines:

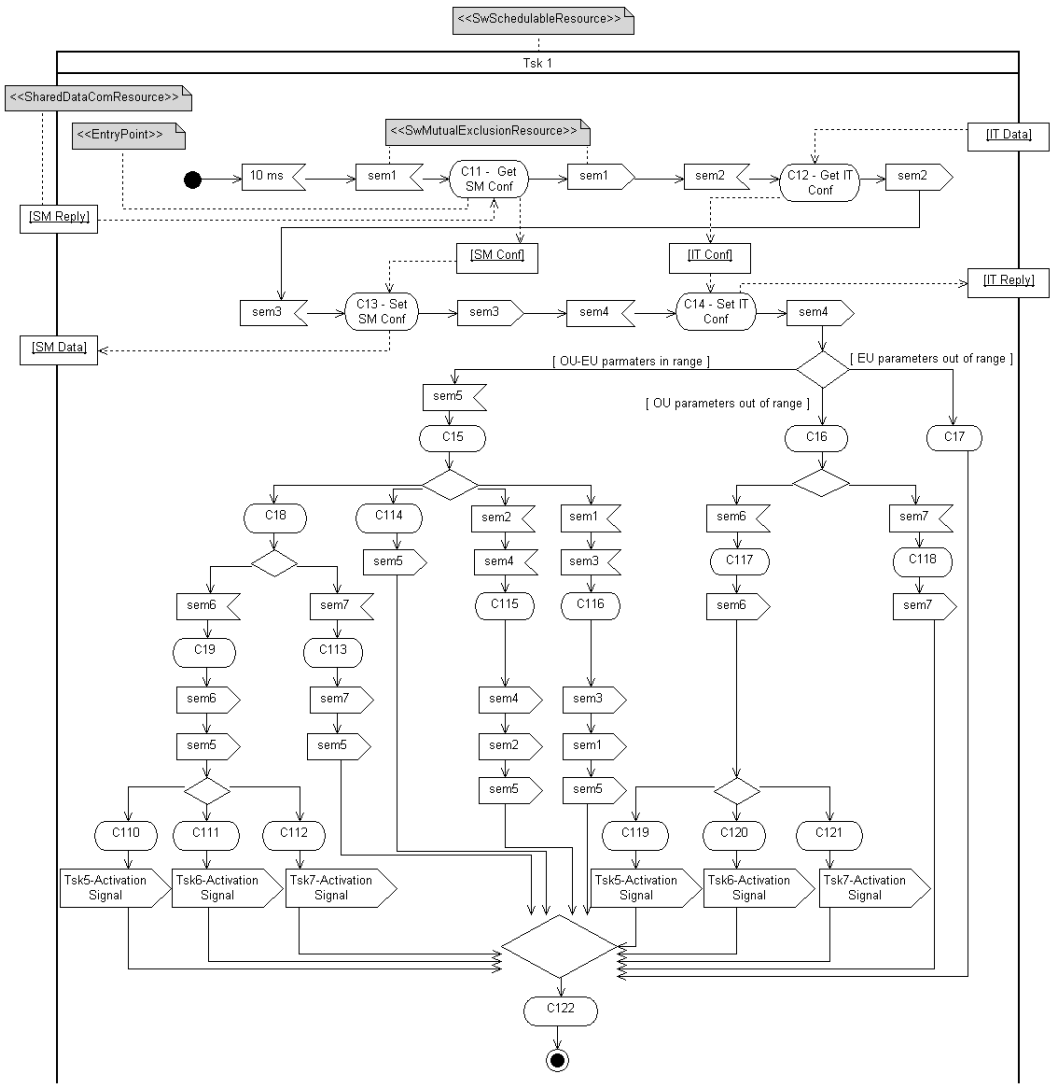

Figure 3.7. SW Design: SDD document. UML-MARTE Activity Diagram of task $Tsk1$ of SC (times expressed in ms). For the sake of readability, only a few UML-MARTE stereotypes, activity names, shared memories, and guard conditions are shown (up to the first branch).

- a task is specified by an open box containing its chunk sequence and decorated with its name and release interval; a down-headed arrow specifies its deadline; and, a double-headed or a single-headed arrow indicates whether the task is recurrent or one-shot, respectively;
- a chunk is specified by a rectangle annotated with its name, Execution Time interval, required resources (e.g. cpu), priority level, and entry-point;
- branches and re-joins in a sequence of chunks are specified using diamonds;
- activations of a one-shot task in reaction to an event thrown by a chunk of a different task are specified by a dotted-arrow from the chunk to the activated task;
- binary semaphore operations are specified by decorating chunks with circles annotated with a sequence of operations, each referred to a semaphore name.

Figure 3.8 exemplifies the concept with reference to the SC task-set of the industrial case study, making explicit the sequence of chunks executed by each task and their semaphore synchronizations. Note that the single schema of Figure 3.8 replaces the Class Diagram of Figure 3.4, the Object Diagram of Figure 3.6, the Activity Diagram of Figure 3.7, and the remaining Activity Diagrams of $Tsk2$ through $Tsk7$ (not reported here).

3.3 Supporting development activities through pTPNs

This Section illustrates how the formal nucleus of pTPNs [26] is used to support design and verification activities of the development process, providing guidance for derivation of pTPN models from timeline schemata to achieve integration with the documentation process prescribed by MIL-STD-498 [112].

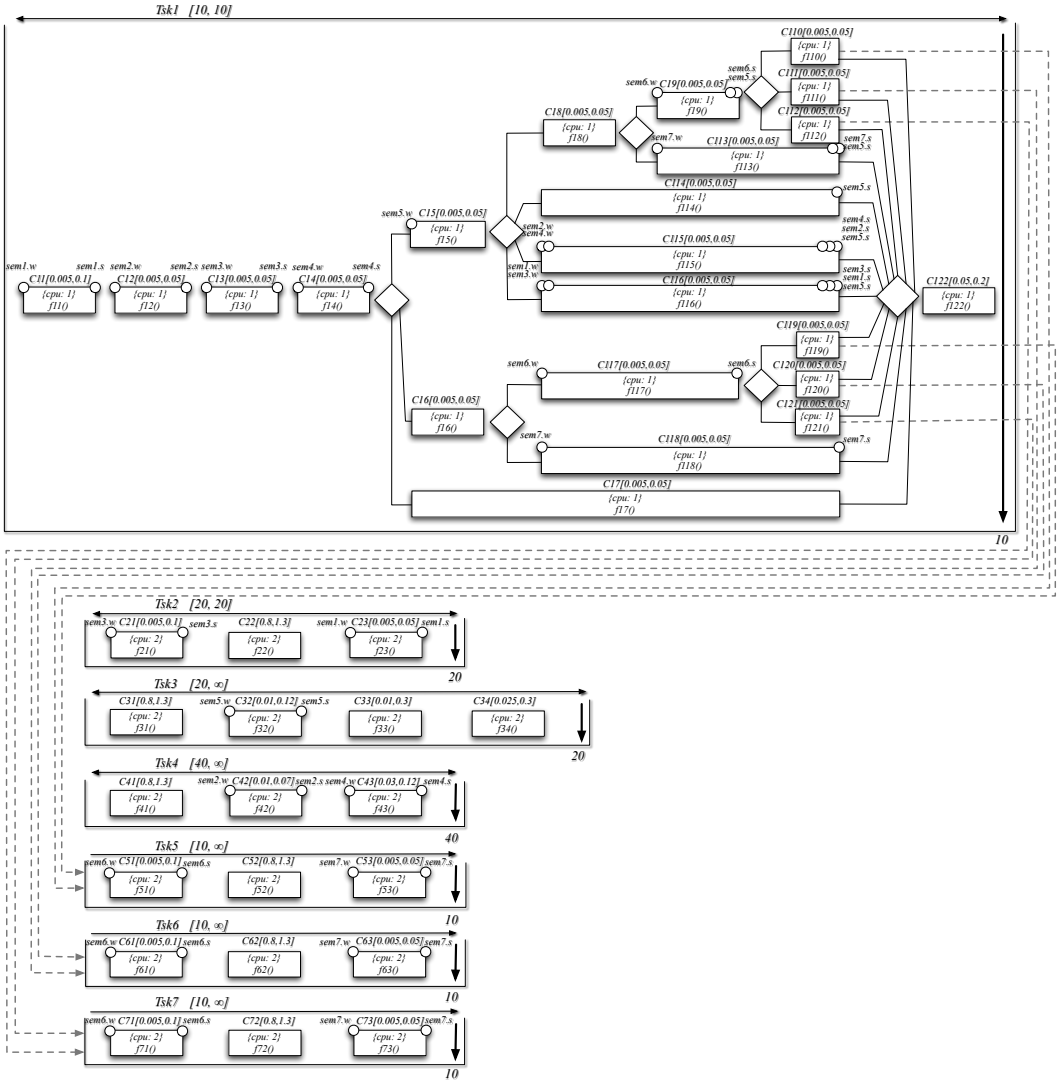

Figure 3.8. SW Design: SDD document. The timeline schema of the SC task-set (times expressed in ms).

It is worth remarking that the translation process can be automated and the resulting pTPN can even remain transparent to the designer, who will be only concerned with the construction of the timeline schema.

In each Subsection an activity, which is an instance of the *Design And Verification Activity* class of the ontological model shown in Figure 1.9, is described. The methodology proposed in this thesis allows to keep trace of their execution, verifying the level of assurance attained in the development process.

3.3.1 Formal specification

A pTPN [26, 25] extends the model of TPNs [78, 13, 118] with a set of preemptable resources whose availability conditions the progress of timed transitions. According to this, each transition is associated with a firing interval, delimited by a static *Earliest Firing Time* (EFT) and a static *Latest Firing Time*, and may request a set of resources with a *priority level*. A transition is *enabled* if all its input places contain at least one token: in this case, it is associated with a dynamic *time-to-fire* taking a non-deterministic value within its static firing interval. An enabled transition is *progressing* and reduces its time-to-fire if every of its associated resources is not requested by any other enabled transition with a higher priority level; otherwise, it is *suspended* and maintains the value of its time-to-fire, which is resumed when the transition is assigned all its associated resources again. A progressing transition is firable if its time-to-fire is not higher than that of any other progressing transition. When a transition fires, a token is removed from each of its input places and a token is added to each of its output places.

Note that the form of syntax and semantics of pTPNs could be reasonably extended so as to account for weights associated with pre-conditions (i.e., arcs from a place to a transition) and post-conditions (i.e., arcs from a transition to a place). In general, this can help in representing contexts where places account for resources and where multiple resources may be needed to perform semaphore actions. However, in the proposed approach, this element of ex-

pressivity is not needed as transitions account for actions that always depend on boolean conditions.

A pTPN model can be derived from a timeline schema either manually or automatically, following a procedure steered by the model structure. In general, the translation associates a place with each logical condition of each job and with each semaphore, and uses transitions to account for job releases, chunk completions, branches, semaphore and priority operations. For the sake of readability, the process is illustrated by referring to the SC task-set of the industrial case study, discussing derivation of the pTPN shown in Figure 3.9 from the timeline depicted in Figure 3.8.

Periodic releases of $Tsk1$ and $Tsk2$ are modeled by transitions with neither input places nor resource requests; therefore, they fire repeatedly with inter-firing times falling within their static firing intervals. According to this, $t10$ and $t20$ model releases of $Tsk1$ and $Tsk2$, respectively. $Tsk3$ and $Tsk4$ are specified as sporadic, since they are both activated by an external event not under scheduler control. However, since SW requirements prescribe the activating events to be periodic, then releases of $Tsk3$ and $Tsk4$ are accounted by transitions $t30$ and $t40$, respectively, with deterministic firing interval of $[20, 20]$ and $[40, 40]$ ms, respectively. One-shot tasks $Tsk5$, $Tsk6$, and $Tsk7$ are activated in a mutually exclusive manner, thus they are mapped on the same pTPN representation. Their releases are represented by transition $t50$, preconditioned by the output place of the transitions that model the completion of the activating chunks. Its firing interval accounts for the time spent in the elaboration of the activating signal.

Job chunks are modeled by transitions with static firing intervals equal to the Execution Time range, with requested resources and static priorities. For instance, transition $t12$ models the completion of chunk $C11$ of $Tsk1$. Branches are modeled by immediate transitions preconditioned by the output place of the preceding chunk; conversely, rejoins are accounted by making the chunks share the same output place. For instance, transitions $t113$, $t146$, and $t157$ are preconditioned by place $p113$ to represent a branch among the mutually exclusive chunks $C15$, $C16$, and $C17$ of $Tsk1$; conversely, transitions

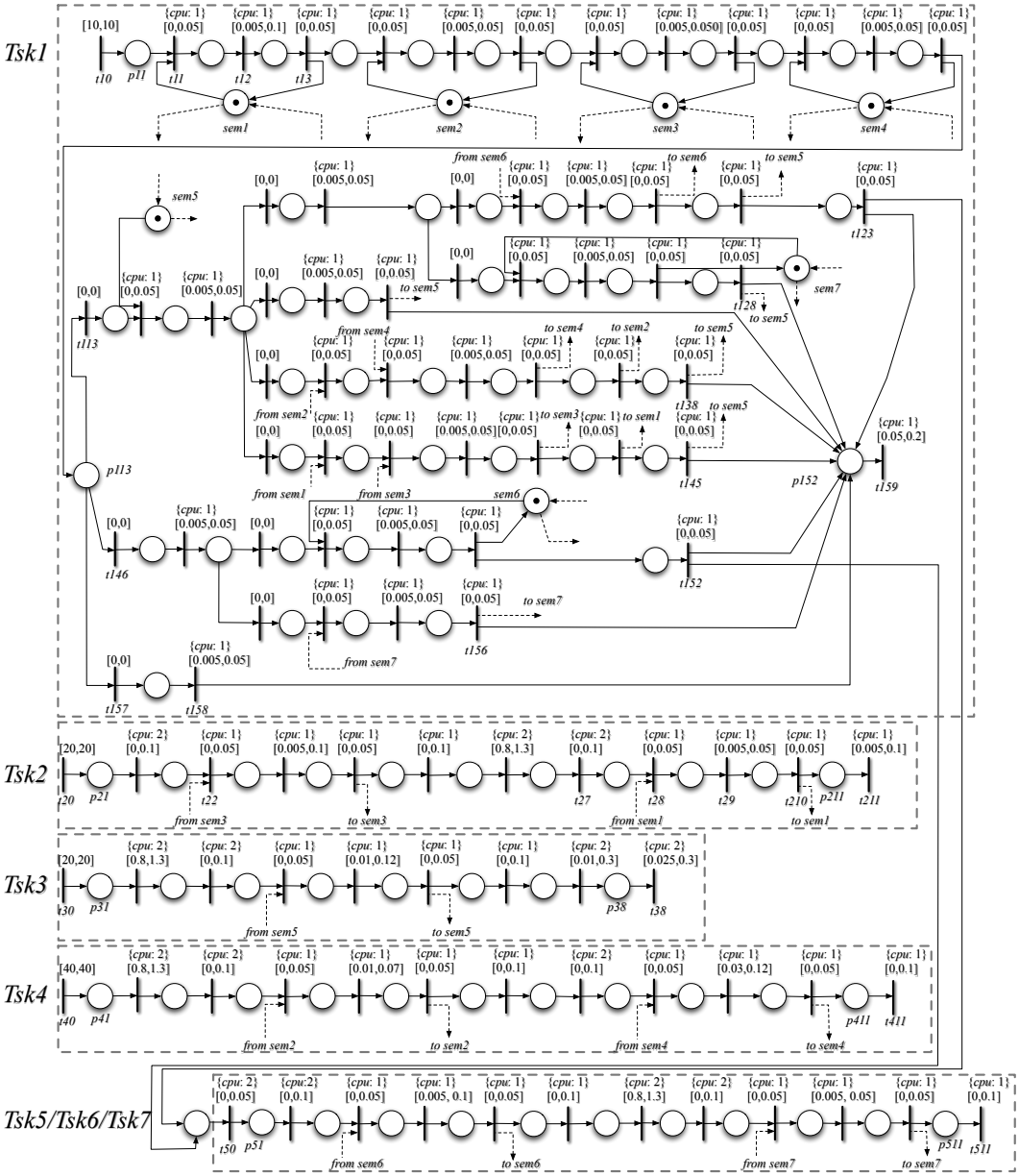

Figure 3.9. SW Design: SDD document. The pTPN model of the dynamic architecture of the SC (times expressed in ms). To reduce the cluttering, the figure does not show the names of places and transitions that are not mentioned in the text.

$t123$, $t128$, $t131$, $t138$, $t145$, $t152$, $t156$, and $t158$ share the output place $p152$ to account for a rejoin after the completion of the mutually exclusive chunks $C17$, $C110$, $C111$, $C112$, $C113$, $C114$, $C115$, $C116$, $C118$, $C119$, $C120$, and $C121$ of $Tsk1$.

According to the priority ceiling emulation protocol [104], low-priority tasks $Tsk2$, $Tsk3$, ..., and $Tsk7$ undergo a priority boost and synchronize on a semaphore in the sections where they access memories shared with the high-priority task $Tsk1$. Binary semaphores are modeled as places initially marked with one token. Since experimental results prove that the time spent in priority boost/deboost and semaphore wait/signal operations is not negligible with respect to the Execution Time range of the SC entry-points, these operations are represented by separate transitions with nonpoint-like firing interval. For instance, $sem1$ represents a binary semaphore synchronizing the access to a memory shared between chunks $C11$ and $C23$; $t27$ models a priority boost; $t11$ and $t28$ account for $sem1$ wait operations; $t12$ and $t29$ represent the completion of $C11$ and $C23$, respectively; $t13$ and $t210$ model $sem1$ signal operations; $t211$ accounts for a priority deboost. This differs from [32], where priority boost and semaphore wait operations are represented by immediate transitions, while priority deboost and semaphore signal operations are accounted by transitions also modeling chunk completions. The abstraction of [32] is motivated by the fact that, on the RTOS in use there, the time spent in priority and semaphore operations is negligible with respect to the Execution Time range of entry-points under development. Thus, since preemption by a different task within the priority ceiling cannot occur at deboost, the model of [32] does not need to distinguish chunk completions from semaphore signal and priority deboost operations.

Note that deadlines do not have a direct counterpart in the pTPN model, although they could be explicitly represented through additional watch transitions as proposed in [14]. However, this would considerably increase the degree of concurrency of the model and thus the complexity of the analysis. Moreover, in most of the cases, deadlines are coincident with minimum inter-release times, so that deadline misses can be easily identified as task releases occurring while

a task-job is still pending.

3.3.2 Architectural verification

The pTPN representation of a task-set opens the way to automated verification of non-functional requirements through state-space analysis. This comprises the step of development which permits to achieve major results, which would be significantly hard to perform without relying on a rigorous formal basis.

Verification of non-functional requirements develops on the enumeration of the space of *state-classes*, which is called *state-class-graph* [26, 25]. A *symbolic run* is a path in the state-class-graph representing the dense variety of runs that execute a sequence of transitions with a dense variety of timings between subsequent firings. Selection and timeliness analysis of all symbolic runs that start with a task release and terminate with its completion, which is called *task symbolic runs*, enable the derivation of the *Best Case Completion Time* (BCCT) and the *Worst Case Completion Time* (WCCT) of each task. This supports verification of deadlines as well as derivation of the minimum laxity which they are attained with.

Architectural verification can be performed through the Oris Tool [24], which implements state-space enumeration, selection of paths attaining specific sequencing and timing constraints, and their tight timeliness analysis. In the case of industrial application, the first round of verification detected a *deadline miss* by one-shot tasks $Tsk5$, $Tsk6$, and $Tsk7$, which are triggered by $Tsk1$. Reduction of Execution Times of chunk entry-points was not feasible, since allocated ranges had already been narrowed up to an acceptable trade-off between *precise estimates* and *safe bounds* [121]. Therefore, the dynamic architecture was redesigned by raising $Tsk1$ period from its initial value of 5 ms up to 10 ms, as shown in the final specification depicted in Figures 3.6 and 3.8. Architectural verification finally yielded the following results: state-space analysis enumerated 4041 state-classes in nearly 1 second; selection and timeliness analysis of task symbolic runs spent nearly 5 seconds to derive 5941, 5660, 5135, 4100, 46 paths for $Tsk1$, $Tsk2$, $Tsk3$, $Tsk4$, $Tsk5/Tsk6/Tsk7$, re-

spectively, with a WCCT of 1.55, 5.67, 4.02, 7.76, 8.56 ms, respectively. This proved that all deadlines were met with minimum laxity of 8.45, 14.33, 15.98, 32.24, 1.44 ms for $Tsk1$, $Tsk2$, $Tsk3$, $Tsk4$, $Tsk5/Tsk6/Tsk7$, respectively.

3.3.3 Disciplined implementation of real-time code

During SW Coding, the proposed approach permits to compile the pTPN model of the dynamic architecture of a CSCI into a skeleton of *control code*, i.e., the code that performs job releases, manages semaphore and priority handling operations, and invokes *functional code* represented by chunk entry-points. The control code conforms with pTPN semantics and can be implemented manually, following a programming discipline steered by the model structure which could be easily automated.

Here translation of pTPN models into real-time code is described, addressing code running on VxWorks 6.5 [122], which comprises a common platform for industrial applications. Each task of the timeline specification can be implemented as a real-time task with a priority and an associated entry-function. In particular, each periodic task is triggered by a periodic alarm and it is actually made recurrent through an explicit loop control structure programmed in its entry-function. At each iteration of the loop, the entry-function synchronizes on the alarm and performs a single job execution. In a similar manner, a loop control structure is also programmed in the entry-functions of sporadic and one-shot tasks. Specifically, at each loop repetition, the entry-function of a sporadic task synchronizes on an external signal, whereas the entry-function of a one-shot task synchronizes on an additional semaphore instrumental to one-shot activation. This semaphore is created by the `init` function and signaled by the activating task.

The architecture of the implementation is further extended to enable observation of possible re-entrant job releases, i.e., the situations in which a job is released before the previous one is completed. Therefore, releases of each task are performed by a single high-priority real-time task that spawns a separate task for each job execution. This keeps the Execution Time of each loop of

the high-priority task sufficiently short to avoid the completion after the subsequent release. Though useful for testing purposes in early implementation stages, this solution is not suitable for deployment code, since the dynamic creation of tasks is deprecated by most regulatory standards for safety-critical SW, e.g., the Ravenscar profile [28].

In the industrial case study, the SC task-set was implemented as a kernel module of VxWorks 6.5 [122]. The `init` function of the kernel module creates semaphores $sem1$, $sem2$, and $sem7$ which are explicitly represented in the timeline schema of Figure 3.8. It also invokes the primitive `sysClkRateSet` to set the period of the system clock equal to the minimum value that can be imposed on the MB, i.e., 1 ms. To obtain fine-grained time measurements, a hardware counter was used that attains 1 ns granularity.

3.3.4 Execution Time profiling

During HW-in-the-loop Testing, the proposed approach supports a disciplined and focused testing that uses the model as an oracle to reveal defects pertaining to concurrency control and timing. In particular, this enables verification of design assumptions about temporal parameters through profiling, with specific emphasis on Execution Times of implemented chunks and timings actually provided by the RTOS. Inconsistencies between assumptions and evidences can be managed through different approaches: by fixing implementation so as to fit specification assumptions; by repeating formal verification on a refined specification that accounts for actually observed parameter values; by providing a recommendation that draws attention on aspects of the implementation that may be not completely covered by formal verification.

The code of a CSCI can be instrumented so as to produce a time-stamped log of each event corresponding to each transition in the pTPN model of the task-set. The impact of logging on a real-time queue is evaluated by estimating its Execution Time through several repetitions of the operation. The operation of logging is conveniently allocated to the dynamic architecture in order to keep instrumentation code separate from functional code of chunk entry-

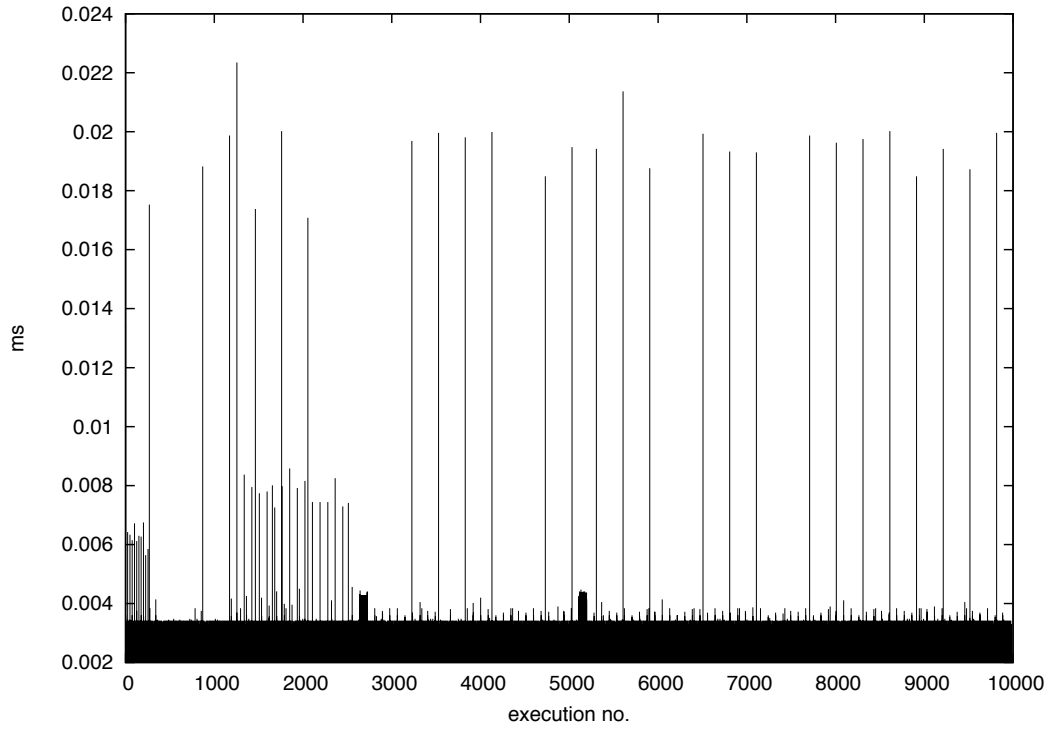

Figure 3.10. Sequence of observed Execution Times of the logging operation.

points. This supports automation of the procedure of code generation, leaving the developer only the responsibility of implementing functional entry-points. At the end of each run of the implementation, the sequence of time-stamped logs is sent to the desktop machine for off-line analysis. Logs support reconstruction of the sequence of states visited during execution, evaluation of the sojourn time in each state, and identification of progressing/suspended transitions in each state. This permits to determine whether the execution log comprises a feasible behavior of the pTPN specification, enabling off-line derivation of the Execution Time of any event as the sum of sojourn times in the visited states where the corresponding transition is progressing. As a salient trait, measurements are carried out by letting chunk entry-points execute in interrupted mode, thus taking into account preemption events, HW/SW interrupts, pipeline and cache effects [121].

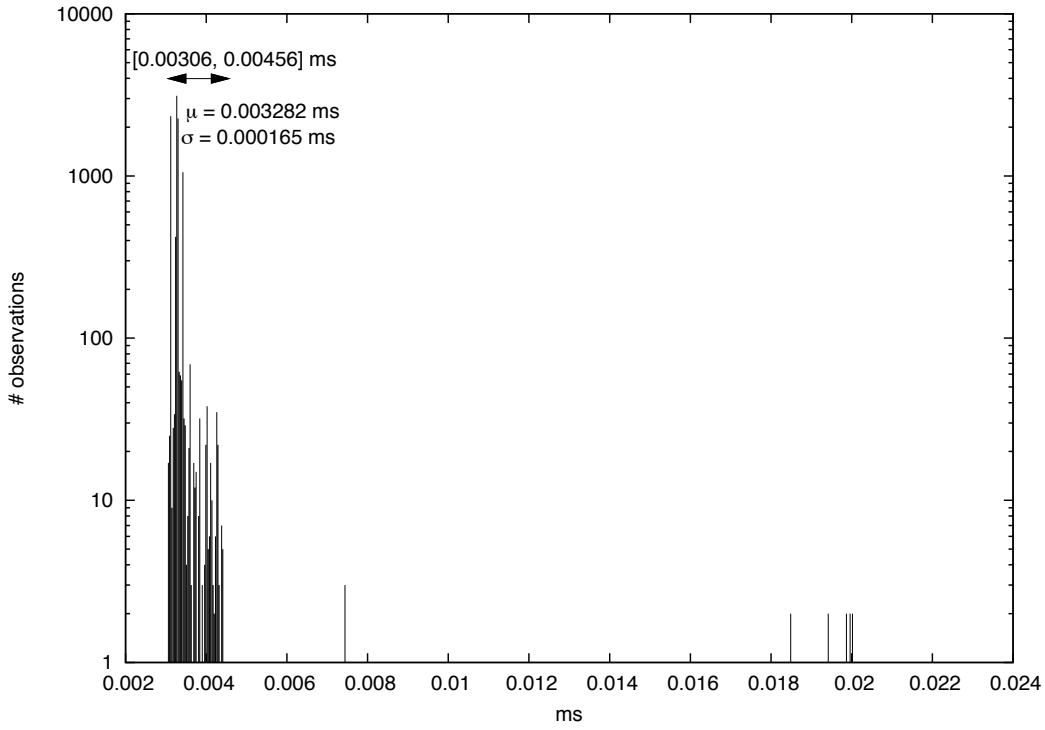

Figure 3.11. Histogram of observed Execution Times of the logging operation.

In the case of industrial application, $10,000$ repetitions of the logging operation are performed and the difference between subsequent logged time-stamps are measured. The sequence and the histogram of observed Execution Times are reported in Figures 3.10 and 3.11, respectively. They show that: 99.5% of the values fall in the range $[0.00306, 0.00456]$ ms, with a mean value of 0.003282 ms and a standard deviation of 0.000165 ms; recurrent peaks in the interval $[0.017, 0.022]$ ms occur in 0.5% of the cases and can be ascribed to timing uncertainties due to HW effects, which are usually in the order of a few tens of μs. Unfortunately, the time spent for logging turned out to be not negligible with respect to the granularity of temporal parameters of the task-set, which in fact are in the order of 0.005 ms to 40 ms. To circumvent overestimation of Execution Times, which may be caused by the logging overhead, firing intervals of temporal parameters were enlarged during iterative

refinements of the dynamic architecture of the task-set.

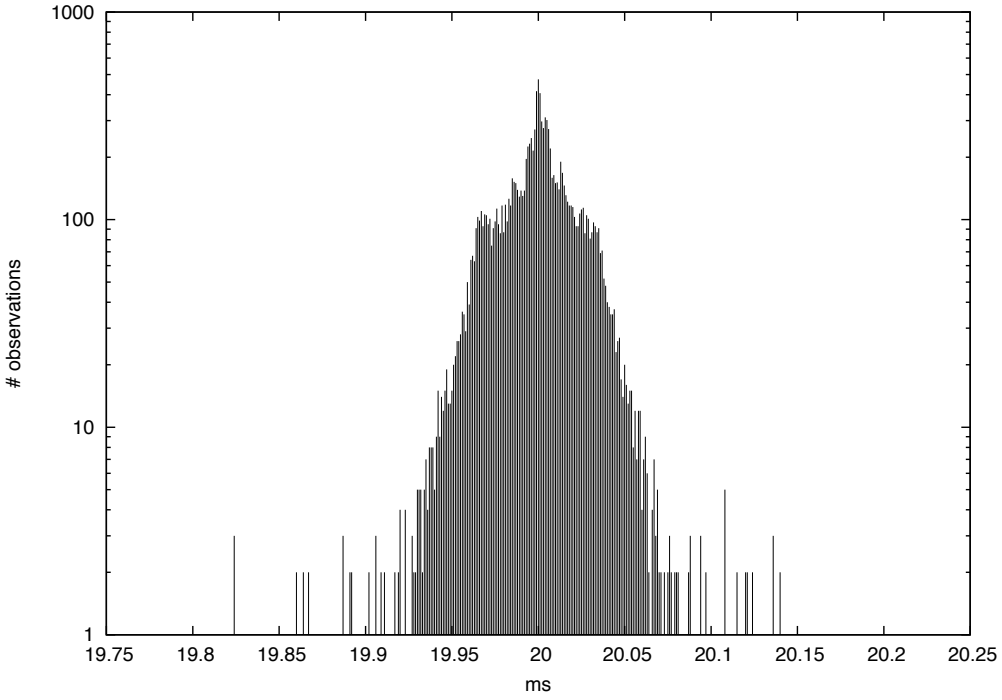

Figure 3.12. Histogram of observed inter-release times of periodic release of $Tsk2$.

In the industrial case study, the SC code was integrated with functional entry-points of its chunks and finally tested in a simulated environment, where selected HCIs/CSCIs of the system (in Figure 3.3, SM, IRC, TVC, LS, and IT) were emulated by a SW application running on a desktop processor connected to the MB through five serial buses. The first round of profiling detected an *unsequenced execution* during which the high-priority task $Tsk1$ was overtaken by the low-priority task $Tsk2$. Inspection of functional code of the two tasks revealed that the failure was caused by a *task programming defect*, consisting of two chunks (i.e., chunk $C21$ of $Tsk2$ and chunk $C13$ of $Tsk1$) synchronizing on a semaphore that was not explicitly represented in the dynamic architecture. The inconsistency was fixed by adding a semaphore named $sem3$ to the SC task-set and by repeating formal verification.

During subsequent executions, *time-frame violations* were detected on dif-

ferent chunks of different tasks. Optimization of chunk entry-points did not succeed in fixing the problem. Finally, the failure was found out to be due to a *cycle stealing* by a VxWorks task named tNetTask, which provides packet-processing network services and runs at priority level 50. The issue was circumvented by refining the model and repeating formal verification. In particular, the priority of SC tasks was raised from their initial values higher than 100, as usual for user tasks, to values lower than 50, as shown in the final specification of Figure 3.9.

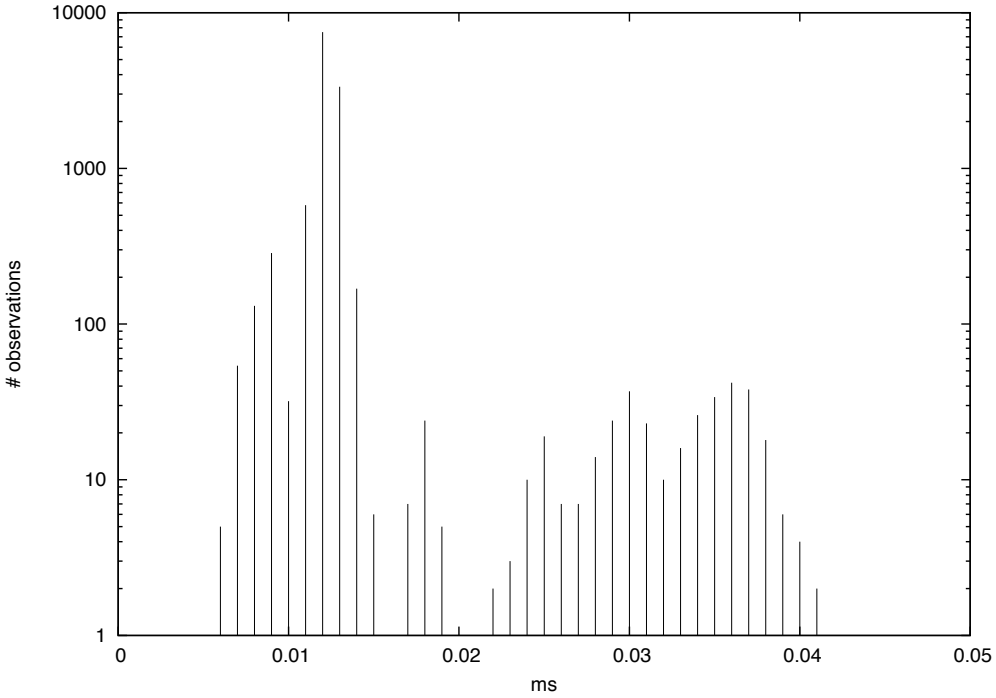

Figure 3.13. Histogram of observed Execution Times of the wait operation performed by $Tsk2$ on semaphore $sem3$.

Inter-release times of periodic task $Tsk2$ is measured, which correspond to inter-firing times of transition $t20$ in the pTPN of Figure 3.9. The n-th inter-release time δ_n is equal to:

$$\delta_n = \big((1+n)\cdot T + \varepsilon_{n+1}\big) - (n\cdot T + \varepsilon_n) = T + \varepsilon_{n+1} - \varepsilon_n, \qquad (3.1)$$

where $T = 20\ ms$ is $Tsk2$ period and ε_n is the duration that elapses between the time $n \cdot T$ at which the n-th task job should be released and the n-th time-stamp. ε_1, ..., ε_N can be assumed to be independent and identically distributed random variables. If ε_1, ..., ε_N were uniformly distributed over an interval $[0, \gamma]$, then δ would be triangularly distributed over $[T - \gamma, T + \gamma]$. However, in the practice, they are not uniformly distributed due to processor, bus, and cache effects. The histogram of observed inter-release times plotted in Figure 3.12 reveals that 98.9% of cases fall within $[19.920, 20.069]\ ms$ with a peak on 20 ms, while the remaining 1.1% fall within $[19.784, 19.920]\ ms$ and $[20.069, 20.217]\ ms$. Fixing the implementation so as to conform with the design assumption of period 20 ms was not a viable option, being release time jitters dependent on the interaction between the RTOS and the MB. Repetition of the analysis on a refined model was not a convenient approach as well, since asynchronous releases largely increase the state space. Therefore, in this case, the most appropriate action seemed to be a warning to subsequent testing stages.

Figure 3.13 shows the histogram of measured Executions Times of the wait operation performed by $Tsk2$ on semaphore $sem3$, which corresponds to transition $t22$ with firing interval $[0, 0.05]\ ms$ in the pTPN of Figure 3.9. Observed Execution Times fall in the interval $[0.006, 0.050]\ ms$, included in the interval $[0, 0.05]\ ms$, with a peak on 0.012 ms. As already remarked in Section 3.3.1, the time spent for semaphore operations on the RTOS in use here is not negligible with respect to the order of the Execution Time of entry-points.

Execution Times of entry-point `f122` of chunk $C122$ of $Tsk1$ is finally measured, which corresponds to transition $t159$ with firing interval $[0.05, 0.2]$ ms in the pTPN of Figure 3.9. The histogram of observed values plotted in Figure 3.14 has a thin spectrum within $[0.080, 0.183]\ ms$, contained in the interval $[0.05, 0.2]\ ms$, with a peak on 0.089 ms. This actually reflects the absence of data-dependent alternatives in the implementation of the entry-point.

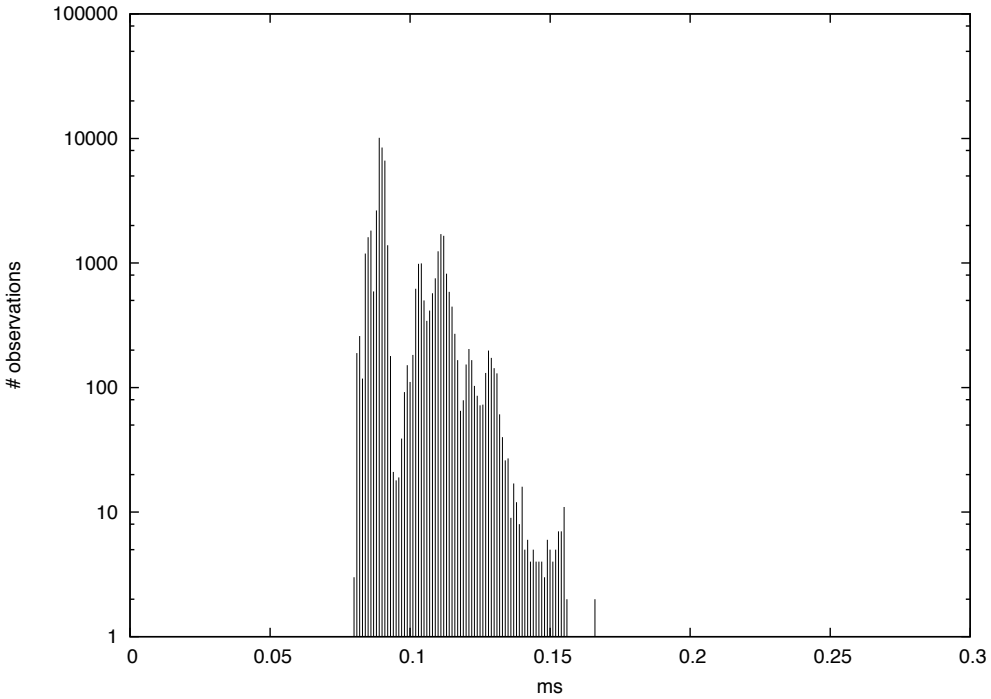

Figure 3.14. Histogram of observed Execution Times of entry-point `f122` of $Tsk1$.

Chapter 4

Implementing a tool to manage the ontology

The ontological abstraction proposed in Section 1.2 can be directly converted into an advanced SW architecture. This has been done by implementing a web application, called *Reliability Availability Maintainability and Safety Engineering Semantics* (RAMSES), built on top of a stack of Semantic Web technologies and standards. In this Chapter the implementation of the tool is described, through its architecture (Section 4.1), its basic and advanced functional capabilities (Section 4.2 and Section 4.3, respectively), and through its experimentation in a real scenario (Section 4.4).

4.1 Architecture and use cases

The web application architecture is shortly sketched in Figure 4.1. The *Presentation Layer* represents the interface between the user and the *Domain Layer*, which realizes, with an object model, the application logic and the data processing functionalities. The *Data Layer* is responsible for data representation

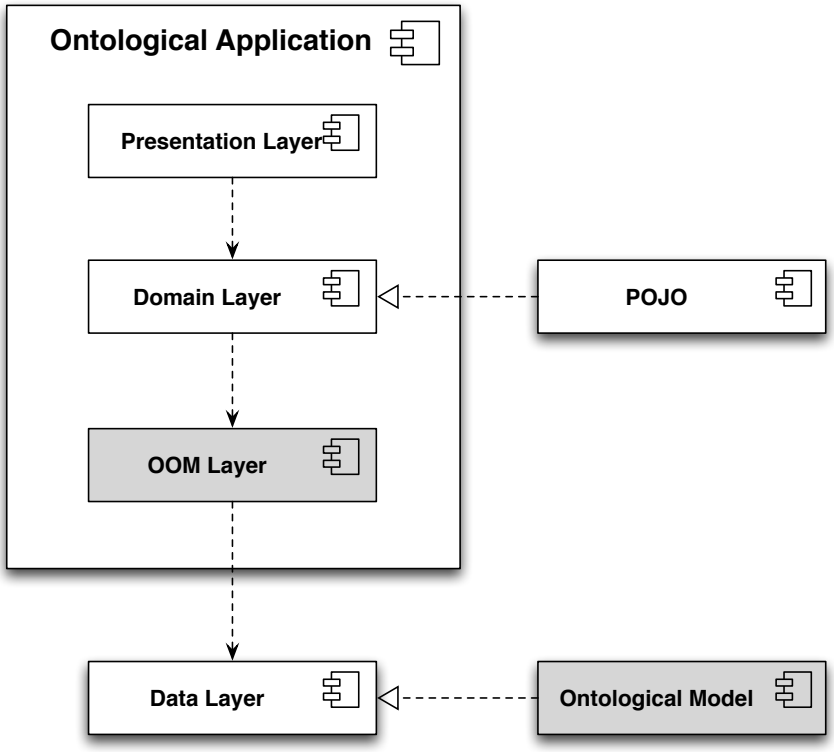

Figure 4.1. Three-tier ontological architecture of a web application incorporating layers interfacing to users (*Presentation Layer*), managing application logic (*Domain Layer*), mapping between object model and data model (*Object to Ontology Mapping Layer*), and realizing data representation and conceptualization (*Data Layer*). The *Domain Layer* is implemented using POJO and the *Data Layer* is realized through an *Ontological Model*.

and conceptualization and is implemented through an ontological model. The *Object-to-Ontology Mapping Layer* bridges the gap between the *Domain Layer* and the *Data Layer* solving the *impedance mismatch*, i.e. the conceptual distance between the object model and the ontological model. In so doing, the domain logic is captured by the ontological model, enabling the generalization of the application logic to adapt it as the concepts describing data change. Figure 4.2 shows that the ontological model is composed by two parts: the one is the *domain model* which is the model described in Section 1.2, the other is the *core model* which contains the entities that allow to perform operations on the domain model, such as adding concepts or properties, through the tool interface without programming.

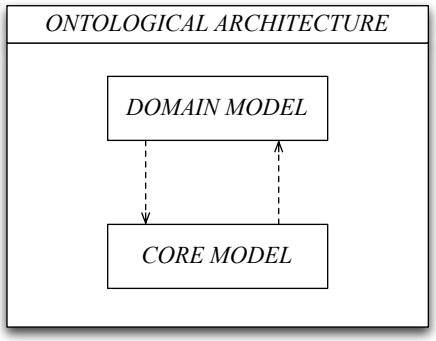

Figure 4.2. The ontological architecture incorporating the *core model* and the *domain model*.

Use of ontologies brings about a number of relevant benefits: i) each project can be represented in OWL form and then it can be exported and imported through this format, enhancing reusability and interoperability; ii) the construction of worksheets and other reports is reduced to the extraction of proper data from the result set generated by a SPARQL query, which in turn is automatically resolved by an ontological reasoner; iii) new information can be inferred from the knowledge base by the ontological reasoner by means, for instance, of predefined SWRL rules.

Figure 4.3 shows the UML Use Case Diagram of the tool. Data is divided in projects and users are divided in groups: a group can work only on predefined

projects. There are four types of users:

- *guest* can only search and view the project data;
- *standard user* can perform *Create, Retrieve, Update, Delete* (CRUD) operations on *Projects* and *Entities* involved in the methodology (e.g. functional requirements, SW components, documents) and other additional functionalities, such as import/export of documents, generation of SW-FTs, visualization of hierarchical views, generation of SW-FMEA worksheets and inspection of required activities and metrics;
- *admin* can perform import/export of a project through its OWL representation and CRUD operations on *User* and *Group*;
- *expert* can perform CRUD operations on resources of the core model, modifying the domain model. He is the only user that has the necessary domain expertise to add new knowledge to the model.

4.2 Basic tool capabilities

The user populates the extensional part of the ontology with data produced along the development life cycle. This enables the execution of basic activities such as the production of SW-FMEA worksheets and SW-FTs and the generation of hierarchical views of structural elements. Classes representing concepts involved in SW-FMEA and SW-FTA are shown in Figure 1.11.

The format of the worksheet depends on standards and practices used in the application context. For instance, the format shown in Figure 4.4 follows the ECSS standard. In other contexts some names of the fields may be slightly different from these; moreover, depending on the stage of the analysis, the tool can generate worksheets in which some fields are omitted.

Part of the concepts contained in the ontology stands for data categories contained in the worksheet. For instance, *Item*, *Functional Requirement*, *Failure Event*, and *Failure Effects* correspond to the classes of the ontology with the

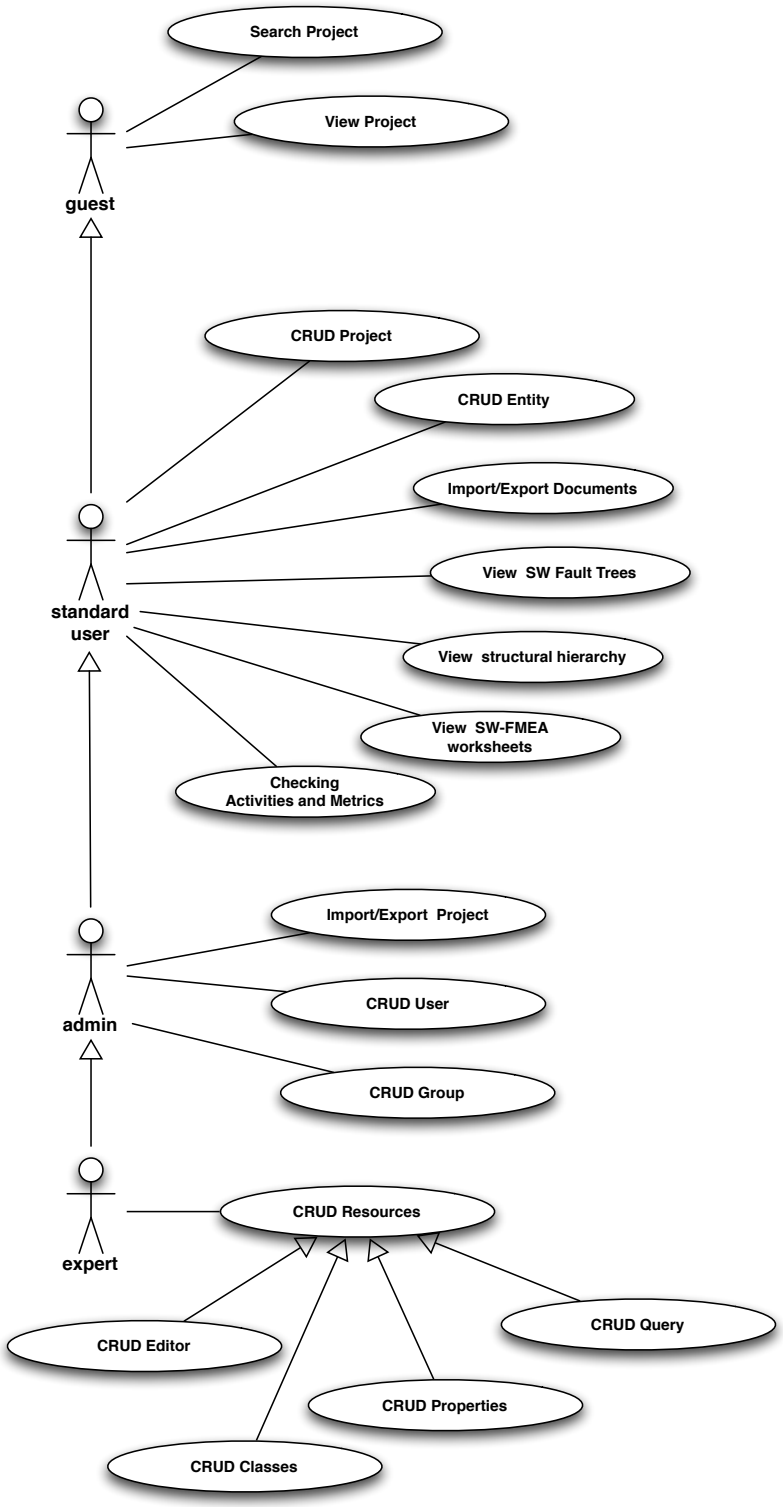

Figure 4.3. Use Case Diagram representing the functionalities supported by RAMSES.

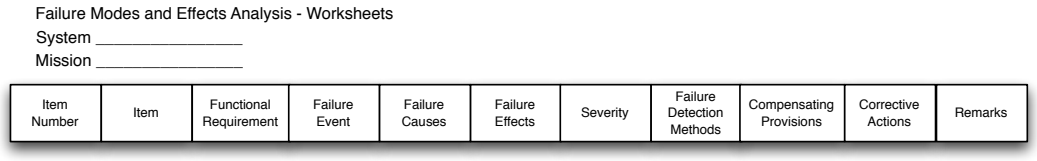

Figure 4.4. The format of a row in the SW-FMEA worksheet, as standardized in ECSS.

same name, *Failure Causes* corresponds to the *Fault* class of the ontology, and *Item Number*, *Severity*, *Detection Methods*, *Compensating Provisions*, *Corrective Actions* and *Remarks* can be added to the model as failure properties. As mentioned in Section 1.1, thanks to a query language as SPARQL, the ontology can be queried to extract the concepts' instances to fill in the worksheet. As far as the ontology is concerned, this is written in OWL and organized as *triples* (or *statements*) in the form of $\langle subject, predicate, object \rangle$, where *subject* is the concept described by the triple, *predicate* describes a relationship between *subject* and *object* which, in turn, is a concept as well. The SPARQL query shown in Listing 4.1 automatically obtains the generic SW-FMEA worksheet of Figure 4.4.

```
SELECT ?idComponent ?component ?functReq ?failureEvent ?failureCause
       ?failureEffect ?severityLevel ?detMethod ?compProv ?corrAct
       ?remarks
WHERE { ?component rdf:type <urn:ramses#SWComponent> .
        ?component <urn:ramses#hasItemId> ?idComponent .
        ?functReq <urn:ramases#isImplementedBy> ?component .
        ?functReq<urn:ramses#hasFailureEvent> ?failureEvent .
        ?fault <urn:ramses#isCausesOf> ?failureEvent .
        ?failureEvent <urn:ramses#hasEffect> ?failureEffect .
        ?failureEvent <urn:ramses#hasSeverityLevel> ?severityLevel .
        ?failureEvent <urn:ramses#hasDetectionMethod> ?detMethod .
        ?failureEvent <urn:ramses#hasCompensatingProvision> ?compProv .
        ?failureEvent <urn:ramses#hasCorrectiveAction> ?corrAct .
        ?failureEvent <urn:ramses#hasRemarks> ?remarks }
```

Listing 4.1. A SPARQL query producing a result set comprising the values for the construction of the SW-FMEA worksheet.

The query that produces SW-FTs is more complex, because the depth

of the tree is not known a priori. Instances of the concepts of the ontological model stand for data needed for the construction of the SW-FTs. For instance, in Figure 4.5 the format of a generic FT is shown: the nodes labelled *Basic Fault*, *OR* and *AND* correspond to subclasses of the *Fault* class, while the TE corresponds to the *Failure Event* class.

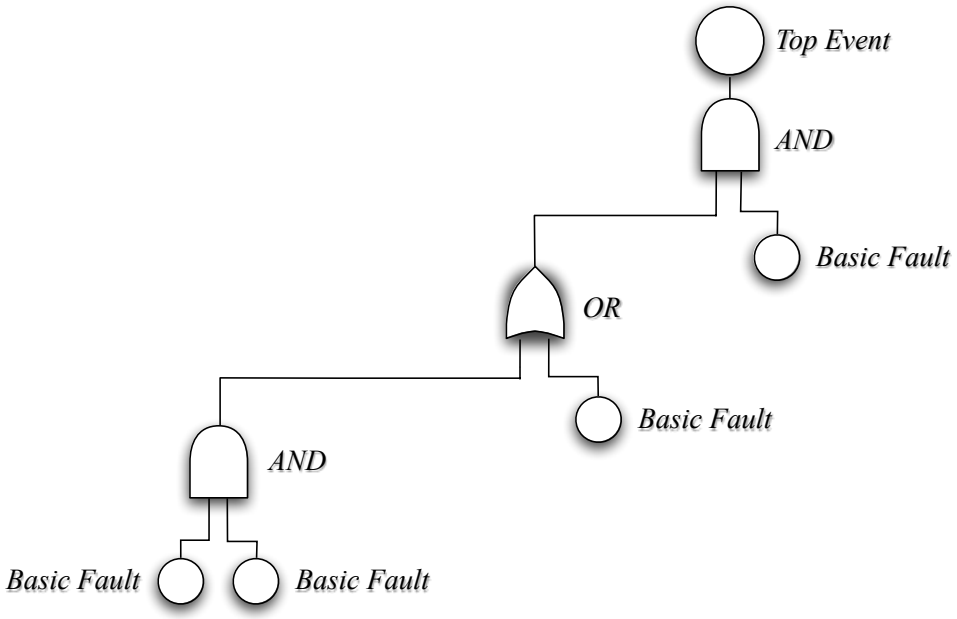

Figure 4.5. The format of a generic FT.

RAMSES can also build hierarchical views of structural elements, providing a clearer picture of the system, by aggregating data scattered in different documents produced along the life cycle. Figure 4.6 shows the hierarchical view of structural elements obtained through the execution of the SPARQL query of Listing 4.2. Thanks to the ontological model, the tool can be adapted to the needs of the application context. In fact, through the tool interface the user *expert* can create his own typical structural hierarchy, modifying the ontological model adding items and editing their properties. The schema shown in Figure 4.6 corresponds to the left side of the model of Figure 1.7.

```
SELECT ?CSCI ?SWComponent ?SWModule ?Method
WHERE { ?CSCI rdf:type <urn:ramses#CSCI> .
        ?CSCI <urn:ramses#hasSWComponent> ?SWComponent .
        ?SWComponent <urn:ramses#hasSWModule> ?SWModule .
        ?SWModule <urn:ramses#hasMethod> ?Method}
```

Listing 4.2. A SPARQL query producing a result set comprising the data used to build the hierarchical view.

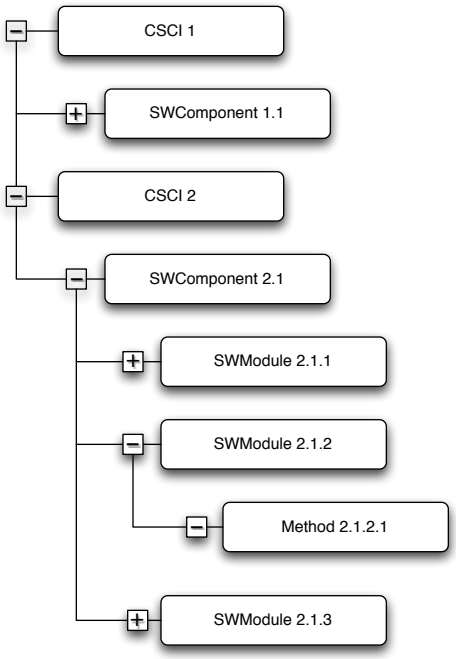

Figure 4.6. Sample of structural hierarchy.

4.3 Advanced tool capabilities

In addition to the previously mentioned capabilities, the tool provides advanced functionalities to ease and improve the development process.

RAMSES aids the analyst in the identification of failure events and supports the accomplishment of testing activities. If a failure event is discovered during the operational phase, the associations between failures, requirements and tests permit to identify the tests that should have covered the failure. Once the analyst has identified the faults that cause the failure and associated

them with structural items (e.g. SW components), other requirements that could be not satisfied are identified by means of the association between SW items and requirements.

The tool can also ease and improve the process of recertification. This can be useful when some changes happen in the development process. These changes may refer to the implementation of SW (i.e. the structural perspective), the requirements (i.e. the functional perspective), or the adopted standard/regulation (i.e. the process perspective). The ontological model reacts to these changes giving evidence of possible inconsistencies arisen among the data inserted in the ontology. The tool also recommends the re-execution of tests or the accomplishment of specific activities so as to conform with a specific standard.

Furthermore, pluggable modules, supporting specific activities, can be devised to produce concepts and associations that, leveraging OWL, can be integrated in the ontology, assuring consistency and coherence with data already present. For instance, a plug-in module implementing the process of tracing requirements, described in Section 2.2.1, has been integrated in the tool. Figure 4.7 shows an UML Activity Diagram describing actions performed by the plug-in module and by RAMSES to obtain instances of the association of Figure 2.2 between requirements and SW components. This is used to verify information contained in the traceability matrix reported in the SDD document. The plug-in is implemented as a Java application which uses AspectC to instrument the scheduler code so as to extract information about SW components executed. The process follows the activities shown in Figure 4.7; the result is an OWL file which is imported in RAMSES to add to the ontology the instances of the association. Other appropriate plug-ins allowing the automatic import of requirements from specification documents can be developed.

In addition, the tool exploits the inference capabilities of an ontological reasoner by means of SPARQL queries or predefined SWRL rules. The reasoning capability is crucial for the verification of the level of assurance. As reported in Section 2.2.3, a requirement is rigorously implemented if the related SW items satisfy a predefined set of predicates. For example, the SWRL

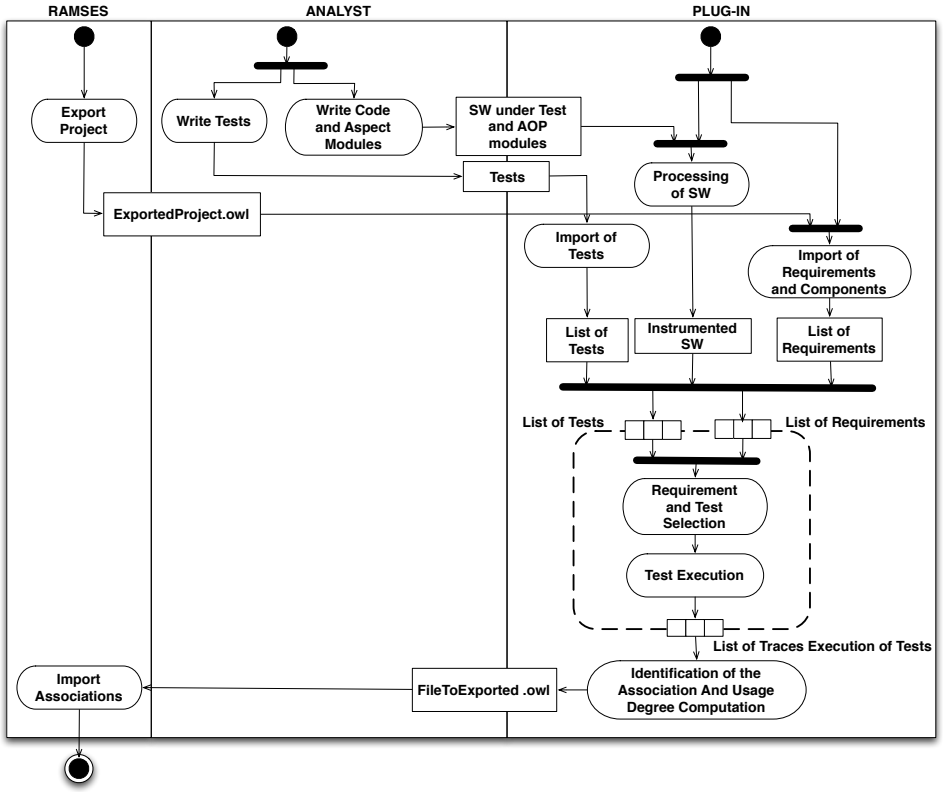

Figure 4.7. UML Activity Diagram showing actions performed by RAMSES and the plug-in module during the activity of tracing requirements.

rule of Listing 4.3 can be used to verify a predicate (an instance of the general form shown at the end of that Section). Thanks to this kind of rules, RAMSES is able to recommend appropriate actions to the analyst, taking advantage of the ontology.

```
ramses:FunctionalRequirement(?r) ∧
∧ ramses:isImplementedBy(?r,?ud) ∧
∧ ramses:hasUsedComponent(?ud,?swc) ∧
∧ ramses:hasSWModule(?swc,?swm) ∧
∧ ramses:hasMethod(?swm,?m) ∧
∧ ramses:hasParamAcc(?m,?spa) ∧
∧ ramses:hasLinkedParameter(?spa, ?sp) ∧
∧ ramses:hasName(?sp,‘‘cyclomatic complexity’’) ∧
∧ ramses:hasParamValue(?spa, ?pv) ∧
∧ swrlb:greaterThan(?pv, 5) ⇒
⇒ ramses:NotRigorous(?f)
```

Listing 4.3. SWRL rule verifying the satisfaction of a predicate: if there exists a SW component `?swc` which is implemented by a SW module `?swm` containing a method `?m` having a McCabe's cyclomatic complexity `?sp` greater than 5 the predicate is violated and the functional requirement is considered not rigorously implemented.

Regarding the SW-FMEA, the tool provides advanced functionalities to ease and improve the execution of the analysis. For example, the tabular nature of the SW-FMEA worksheet leads to a scattered representation of information about the system. Therefore, the search for the failures causing a certain effect compels the analyst to jump from one row to another of the worksheets, searching for the content of cells labelled *Failure Effects* (Figure 4.4). On the contrary, the ontology of Figure 1.8 makes explicit the association between a *Failure Event* and a *Failure Effect*, simplifying the search for the failures causing a certain effect. In fact, instances of the required entities can be easily retrieved through a SPARQL query. In other words, the ontology provides the tool the capability of gathering information which is hidden (e.g. scattered throughout the worksheets) in a classical SW-FMEA process.

4.4 Practical experimentation on a real case study

The major application of the tool has been in the AASTR project, a space project managed at the FinMeccanica site of Selex Galileo in Florence. The project targeted the development of the *Active Pixel Sensor (APS) Autonomous Star Tracker* for the Bepi Colombo Mission under the control of *Astrium Space Deutschland* (ASD) and *European Space Agency* (ESA). The main purpose of the AASTR SW is acquiring data from the APS detector, performing the star clustering and filtering, and then calculating the attitude and the angular rate. Attitude propagation allows tracking of the stars. To give an idea of our case study, AASTR SW is composed by a single CSCI, having 8 possible operation modes and 243 functional requirements.

To avoid disclosure of classified details, in the following the tool is described through the report of its application to a smaller project concerning a scheduler of an electromechanical system for immunoenzymatic analyses, manufactured by BioMérieux, a worldwide group specialized in the field of in vitro diagnostics for medical and industrial applications. The scheduler [99] has been developed in the Software Technologies Laboratory (University of Florence). The system executes multiple concurrent analyses, the aim of the scheduler is to minimize the completion time for the overall set of analyses, avoiding conflicts in the shared hardware. The scheduler is composed by a single CSCI and has to satisfy 7 functional requirements; one of them, called *Constraints loading*, imposes timing constraints to analyses execution. The CSCI is made up of 10 SW components, 10 SW modules, and 76 methods.

The user creates a project through the tool interface shown in Figure 4.8. The project is associated with a standard, which is an instance of the class *Applicable Regulation*, and with some structural items. The user can insert information about the items composing the system and the related functional requirements, with their failure events, the structural faults and the failure effects; all of them are loaded in the ontological base through the tool interface. In Figure 4.9 properties of a CSCI are shown.

Once the previous data have been input, the associations between failure

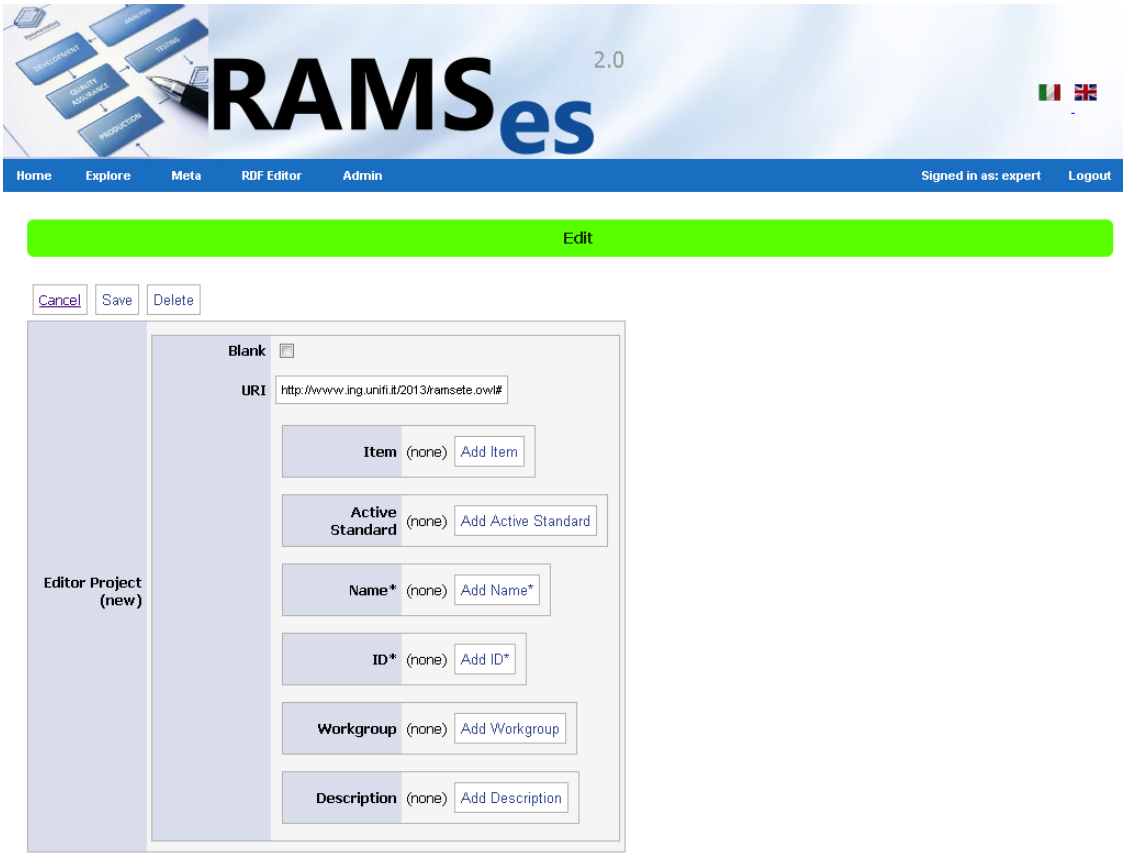

Figure 4.8. Screenshot showing the tool interface provided to create a new *Project*.

events and effects discussed in Section 4.3 can be visualized, helping the analyst in focusing the causes of a certain effect.

Collected structural information (i.e. instances of *SW Component*, *SW Module*, *Method*), added to the ontological base, can be browsed through the tool interface. Hierarchical views such the one generated by the query of Listing 4.2, can be displayed as in Figure 4.10. Furthermore, the SW-FMEA worksheets and the SW-FTs related to the failure events inserted in the ontology can be automatically generated.

To verify the association of requirements with structural items the plug-in described in Section 2.2.1 can be executed. The application of the plug-in to the scheduler has shown that, for instance, the functional requirement

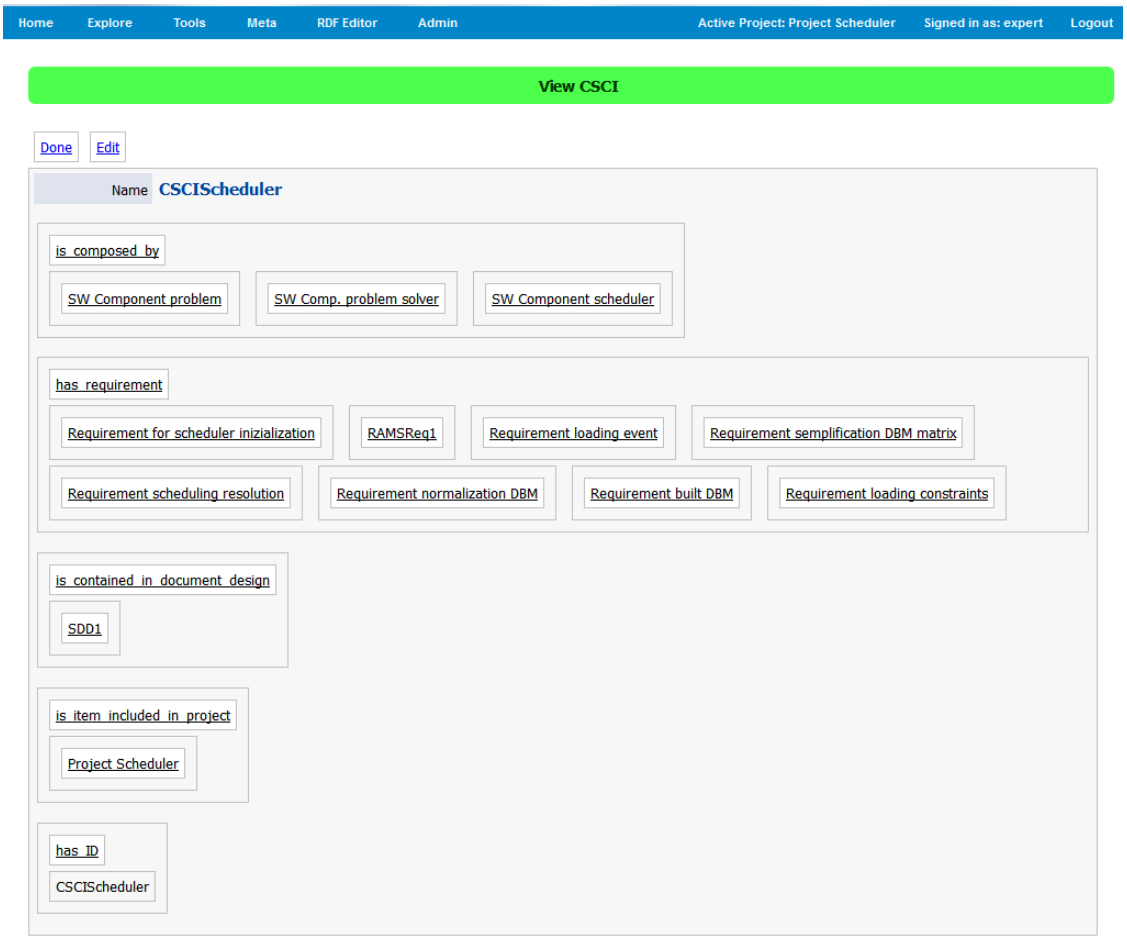

Figure 4.9. Screenshot showing the tool interface provided to show information about an instance of a *CSCI*.

Constraints loading is implemented by 5 SW components whose usage degrees are as follows:

SW Component	Usage Degree (%)
block	100.00
matrix	42.86
problem	60.00
problem_solver	10.00
startingjitter_strategies	16.67

This particular result verifies that the actual implementation is compliant with

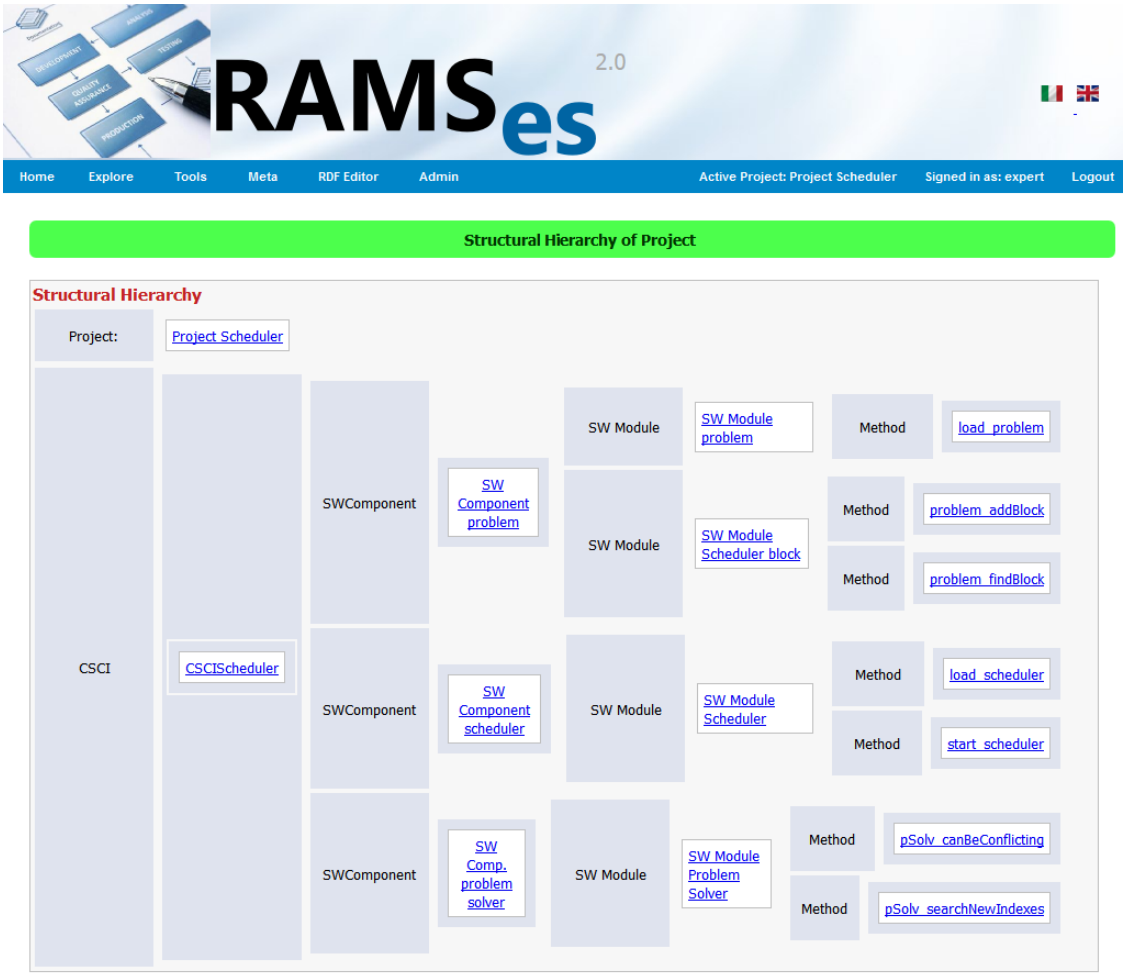

Figure 4.10. The generated system hierarchical view as generated by the RAMSES tool through a SPARQL query on the ontology.

the traceability matrix reported in the documentation of the scheduler, thus witnessing the goodness of the approach.

In the project, 24 metrics have been used. Though only the following metrics have been considered: *McCabe cyclomatic complexity*, *number of executable statements*, *number of lines of code*, *number of lines of comments*. In any case, new metrics can be dynamically added at any time through the tool interface. As the values of the metrics and of the executed development activities are inserted in the ontological base, the reasoner controls if the required

level of assurance for a requirement is reached by the current implementation using a rule such that of Listing 4.3. If a violation is detected by the reasoner, RAMSES indicates the values of code metrics that are not compliant with the prescriptions (Figure 4.11) and the development activities that are executed with not standard values (Figure 4.12).

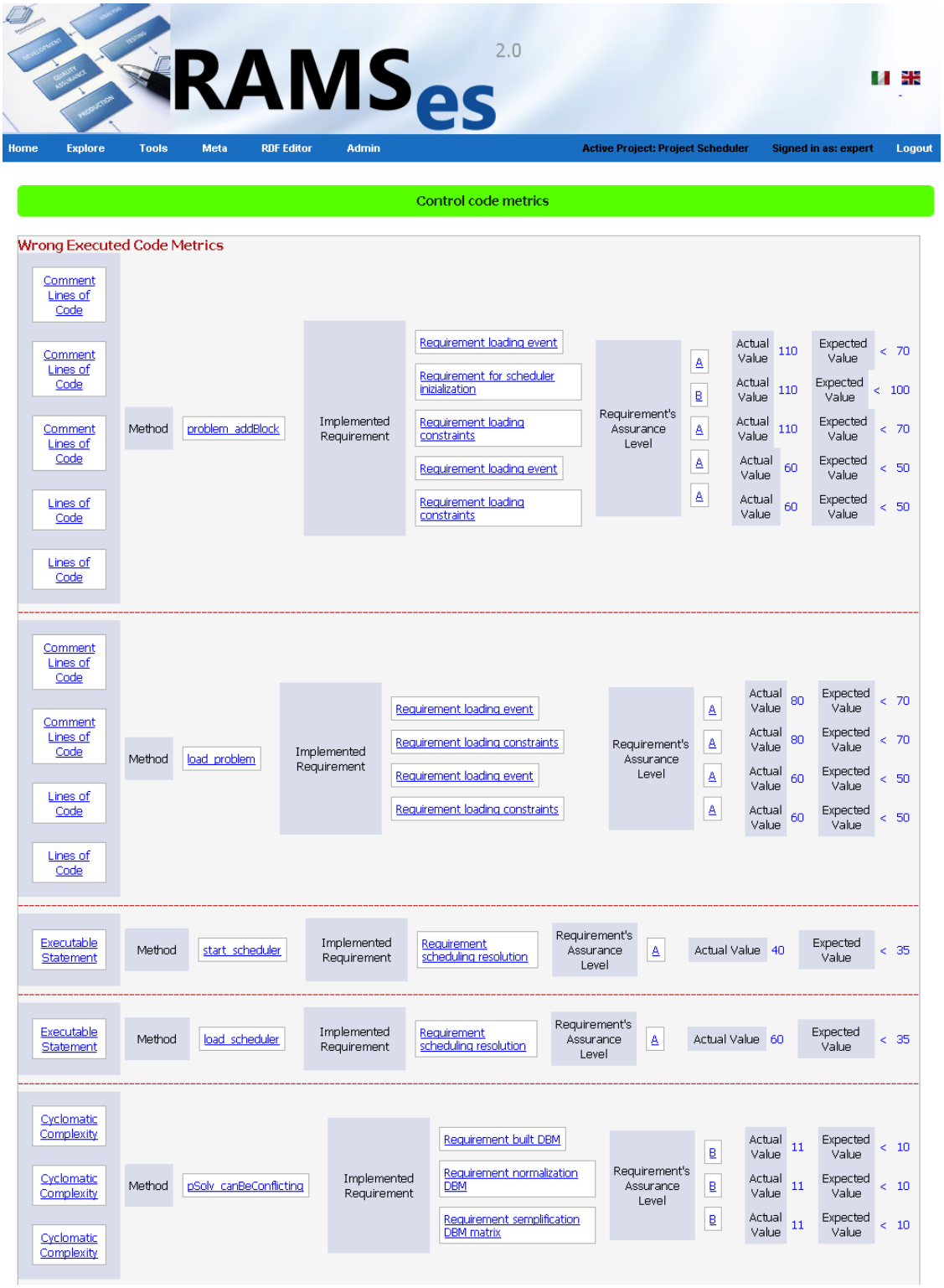

Figure 4.11. Screenshot of the RAMSES tool showing code metrics with not compliant values.

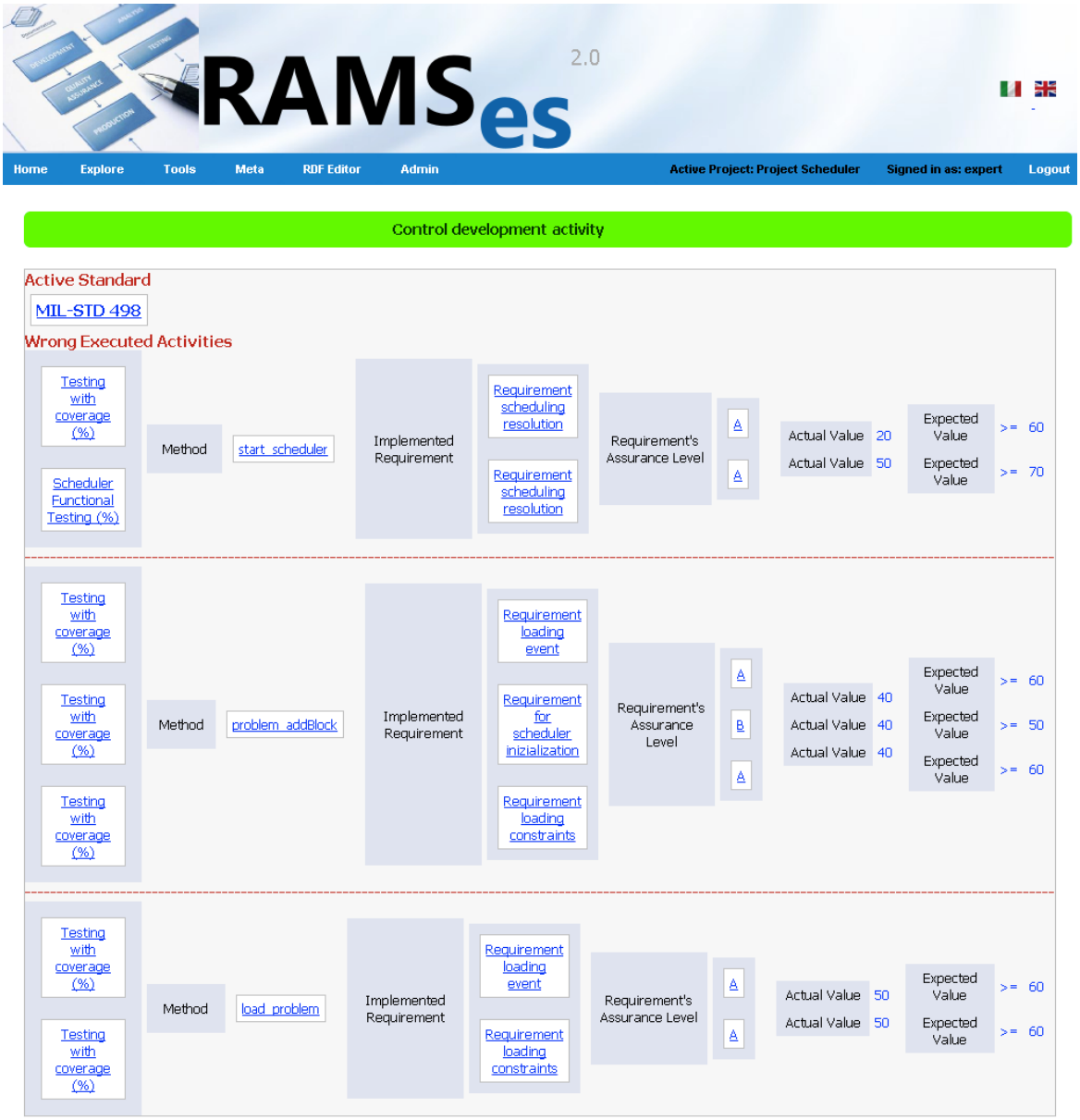

Figure 4.12. Screenshot of the RAMSES tool showing activities executed with not compliant values.

Conclusions

This thesis proposes an ontological model formalizing concepts and data involved in the development process of safety-critical systems, giving them a precise semantics, so as to integrate in a common framework the activities performed and the outcomes produced during the whole life cycle. The work presented in [16] and [15] is enhanced by adding the process perspective to the structural and functional ones, capturing concepts involved in the regulation of industrial processes. The proposed methodology is based on an ontological formalization of the development process which is robust enough to enforce cohesion and consistency among information acquired along different phases of the development process, possibly contributed by different parties. In this manner, a general framework is obtained that can be adapted to any given context. The framework can be tailored to different regulatory standards leveraging the extensibility and the manageability provided by the ontological architecture.

Furthermore, the methodology includes a comprehensive approach for managing the documentation process prescribed by MIL-STD-498 through UML-MARTE diagrams and for supporting design and verification activities through the formal nucleus of pTPNs. UML-MARTE is conveniently used to manage the documentation process prescribed by MIL-STD-498, providing a semi-formal specification that is practical enough to fit the industrial practice and sufficiently structured to enable subsequent application of advanced formal

methods. This provides an effective ground for deployment of pTPN theory, supporting the steps of design, implementation, and verification.

The formalized ontological conceptualization enables effective application of reasoning tools aiding the accomplishment of crucial and effort-expensive activities. This includes basic functionalities such as efficient production of SW-FMEA worksheets and SW-FTs as well as hierarchical views of structural elements. The ontological formalization also provides the ground for advanced functionalities such as the verification of the execution of prescribed activities so as to conform to a specific standard and the automatic identification of SW elements impairing the required quality of service.

The ontological model opened the way to the implementation of a tool, called RAMSES, built on top of well-established Semantic Web technologies. RAMSES automates the processing of data by exploiting query capabilities of SPARQL as well as inference capabilities of an off-the-shelf reasoner. The tool can be adapted to different industrial processes/life cycles, leveraging the adaptability of the ontological model. Furthermore modules performing specific activities, can be plugged into the tool to produce concepts and associations that, leveraging OWL, can be integrated in the ontology, assuring consistency and coherence with data already present. For instance, a specific plug-in was devised and the information generated by its execution was integrated in the ontology to verify the consistency of documents produced along the development life cycle.

The tool has been experimented in a satellite star tracker and in a scheduler of an electromechanical system performing biological analyses; some results are reported from the latter. The experimentation has proved feasibility and effectiveness of both the ontological approach and the tool, showing improvements over current practices.

Bibliography

[1] R. Alur, I. Lee, and O. Sokolsky. Compositional refinement for hierarchical hybrid systems. In *Hybrid Systems: Computation and Control, LNCS 2034*, pages 33–48. Springer–Verlag, 2001.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11 – 33, January 2004.

[3] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Trans. on SW Eng.*, 30(5):295 – 310, May 2004.

[4] S. Balsamo, M. Marzolla, and R. Mirandola. Efficient performance models in component-based software engineering. In *Proceedings of the* 32^{nd} *EUROMICRO Conf. on SW Eng. and Advanced Applications*, pages 64–71, Washington, DC, USA, 2006. IEEE Computer Society.

[5] K. Beck and W. Cunningham. A laboratory for teaching object oriented thinking. *SIGPLAN Not.*, 24(10):1–6, 1989.

[6] S. Beckera, H. Koziolekb, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and SW*, 82:3–22, 2009.

[7] S. Bernardi, J. Campos, and J. Merseguer. Timing-failure risk assessment of UML design using Time Petri Net bound techniques. *IEEE Transactions on Industrial Informatics*, 7(1):90–104, February 2011.

[8] S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable Petri Net models. In *Proceedings*

of the 3rd international workshop on Software and performance, WOSP '02, pages 35–45, New York, NY, USA, 2002. ACM.

[9] S. Bernardi and J. Merseguer. Performance evaluation of UML design with Stochastic Well-formed Nets. *Journal of Systems and Software*, 80:1843–1865, 11 2007.

[10] S. Bernardi, J. Merseguer, and D. C. Petriu. Dependability modeling and analysis of software systems specified with UML. *ACM Computing Survey*, 2011.

[11] M. Bernardo, L. Donatiello, and P. Ciancarini. Stochastic process algebra: From an algebraic formalism to an architectural description language. In *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pages 173–182. Springer Berlin / Heidelberg, 2002.

[12] T. Berners-Lee. *Semantic web roadmap.* http:// www.w3.org/2001/sw, 1998.

[13] B. Berthomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. on SW Eng.*, 17(3):259–273, March 1991.

[14] B. Berthomieu and M. Menasche. An enumerative Approach for Analyzing Time Petri Nets. In R. E. A. Mason, editor, *Information Processing: Proceedings of the IFIP Congress 1983*, volume 9, pages 41–46. Elsevier Science, 1983.

[15] I. Bicchierai, G. Bucci, C. Nocentini, and E. Vicario. Integrating metrics in an ontological framework supporting SW-FMEA. In *3rd International Workshop on Emerging Trends in Software Metrics, WETSoM 2012*, pages 35 –41, 2012.

[16] I. Bicchierai, G. Bucci, C. Nocentini, and E. Vicario. An ontological approach to systematization of SW-FMEA. In *Proceedings of the 31st Int. Conf. on Computer Safety, Reliability, and Security*, SAFECOMP'12. Springer-Verlag, 2012.

[17] P. Bieber, R. Delmas, and C. Seguin. DALculus: theory and tool for development assurance level allocation. In *Proceedings of the 30th International Conference on Computer Safety, Reliability, and Security*, SAFECOMP'11, pages 43–56. Springer-Verlag, 2011.

[18] C. Bodenstein, F. Lohse, and A. Zimmermann. Executable specifications for model-based development of automotive software. In *IEEE International Conference on Systems, Man, and Cybernetics, SMC 2010*, pages 727 –732, October 2010.

[19] H. Boley, T. Athan, A. Paschke, S. Tabet, B. Grosof, N. Bassiliades, G. Governatori, F. Olken, and D. Hirtle. *Specification of Deliberation RuleML 1.0*. http://wiki.ruleml.org/index.php/Specification, 2012.

[20] J.B. Bowles and C. Wan. Software Failure Modes and Effects Analysis for a small embedded control system. In *Reliability and Maintainability Symposium, 2001. Proceedings. Annual*, pages 1 –6, 2001.

[21] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/2008/REC-xml-20081126, November 2008.

[22] D. Brickley and R.V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. http://www.w3.org/TR/rdf-schema, February 2004.

[23] BSI - British Standard Institution. BS5760 Reliability of Systems, Equipment and Components Part 5. Guide to Failure Modes, Effects and Criticality Analysis (FMEA and FMECA). Technical report, British Standard Institution, 1991.

[24] G. Bucci, L. Carnevali, L. Ridi, and E. Vicario. Oris: a Tool for Modeling, Verification and Evaluation of Real-Time Systems. *Int. Journal of Software Tools for Technology Transfer*, 12(5):391 – 403, 2010.

[25] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Modeling Flexible Real Time Systems with Preemptive Time Petri Nets. In *Proc. of the* 15^{th} *Euromicro Conf. on Real-Time Systems (ECRTS03)*, July 2003.

[26] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Timed State Space Analysis of Real Time Preemptive Systems. *IEEE Trans. on SW Eng.*, 30(2):97–111, February 2004.

[27] G. Bucci, V. Sandrucci, and E. Vicario. An Ontological SW Architecture Supporting Agile Development of Semantic Portals. In *Software and Data Technologies*, volume 22 of *Communications in Computer and Information Science*, pages 185–200. Springer Berlin Heidelberg, 2009.

[28] A. Burns, B. Dobbing, and T. Vardanega. Guide on the use of the ADA Ravenscar profile in high integrity systems. *ADA Letters*, XXIV(2):1–74, 2004.

[29] G. Buttazzo. *Hard Real-Time Computing Systems*. Springer, 2005.

[30] BWB - Federal Office for Military Technology and Procurement of Germany. *V-Model 97, Lifecycle Process Model-Developing Standard for IT Systems of the Federal Republic of Germany. General Directive No. 250*, June 1997.

[31] Byteworx. *Byteworx FMEA official website*. http://www.byteworx.com.

[32] L. Carnevali, L. Ridi, and E. Vicario. Putting preemptive Time Petri Nets to work in a V-Model SW life cycle. *IEEE Trans. on SW Engineering*, 37(6), November/December 2011.

[33] F. Cassez and K. G. Larsen. *The Impressive Power of Stopwatches*, volume 1877. LNCS, August, 2000.

[34] CENELEC. *EN 50128 - Railway applications: SW for railway control and protection systems*, 1997.

[35] CENELEC. *EN 50126 - Railway applications: the Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*, 2007.

[36] D. Compare, A. D'Onofrio, A. Di Marco, and P. Inverardi. Automated Performance Validation of Software Design: An Industrial Experience. In *ASE*, pages 298–301, 2004.

[37] V. Cortellessa and R. Mirandola. PRIMA-UML: a performance validation incremental methodology on early UML diagrams. *Science of Computer Programming*, 44:101–129, July 2002.

[38] Dept. of Aerospace Engineering - Polytechnic of Milan. *RTAI: Real Time Application Interface for Linux*. https://www.rtai.org.

[39] Die Beauftragte der Bundesregierung für Informationstechnik. *V - ModelXT - Definition and documentation on the web*.

[40] S. Distefano, M. Scarpa, and A. Puliafito. From UML to Petri Nets: The PCM-Based Methodology. *IEEE Trans. on SW Engineering*, 37(1):65 –79, January/February 2011.

[41] L. Dittmann, T. Rademacher, and S. Zelewski. Performing FMEA Using Ontologies. In *Proceedings of 18th International Workshop on Qualitative Reasoning (QR04)*, pages 209–216, Northwestern University, Evanston, USA, August 2004.

[42] I. M. Dokas and C. Ireland. Ontology to support knowledge representation and risk analysis for the development of early warning system in solid waste management operations. In *Int. Symp. on Environmental Software Systems (ISESS) 2007*, 2007.

[43] M. Eaddy, A. Aho, and G. C. Murphy. Identifying, assigning, and quantifying crosscutting concerns. In *Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques*, ACoM '07, Washington, DC, USA, 2007. IEEE Computer Society.

[44] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Trans. on SW Eng.*, 29:210–224, March 2003.

[45] European Cooperation for Space Standardization. *ECSS-E-ST-40C Space Engineering - Software*, March 2009.

[46] European Cooperation for Space Standardization. *ECSS-Q-ST-30-02C Space product assurance - Failure modes, effects (and criticality) analysis (FMEA/FMECA)*, March 2009.

[47] European Cooperation for Space Standardization. *ECSS-Q-ST-80C Space product assurance - Software product assurance*, March 2009.

[48] D. C. Fallside and P. Walmsley. *XML Schema Part 0: Primer Second Edition.* http://www.w3.org/TR/xmlschema-0, February 2004.

[49] M.S. Feather. Towards a unified approach to the representation of, and reasoning with, probabilistic risk information about software and its system interface. In *15th International Symposium on Software Reliability Engineering. ISSRE 2004*, pages 391 – 402, Nov. 2004.

[50] Q. Feng and R. R. Lutz. Bi-directional safety analysis of product lines. *Journal of Systems and Software*, 78(2):111 – 127, 2005.

[51] A. Fiaschetti, F. Lavorato, V. Suraci, A. Palo, A. Taglialatela, A. Morgagni, R. Baldelli, and F. Flammini. On the use of semantic technologies to model and control security, privacy and dependability in complex systems. In *Proceedings of the 30th international conference on*

Computer safety, reliability, and security, SAFECOMP'11, pages 467–479, Berlin, Heidelberg, 2011. Springer-Verlag.

[52] Freescale Semiconductor. *MPC5200B Data Sheet*, 2010.

[53] P.L. Goddard. Software FMEA techniques. In *Reliability and Maintainability Symposium, 2000. Proceedings. Annual*, pages 118 –123, 2000.

[54] V. Grassi and R. Mirandola. PRIMAmob-UML: a methodology for performance analysis of mobile software architectures. In *Proc. of the* 3^{rd} *Int. Workshop on SW and Performance*, WOSP '02, pages 262–274, New York, NY, USA, 2002. ACM.

[55] V. Grassi, R. Mirandola, and A. Sabetta. Filling the gap between design and performance/reliability models of component-based systems: a model driven approach. *Journal of Systems and SW*, 80, 2007.

[56] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[57] E. J. Henley and H. Kumamoto. *Reliability Engineering and Risk Assessment*. Englewood Cliffs, N.J.: Prentice Hall, 1981.

[58] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. *Proc. of the IEEE*, pages 84–99, 2003.

[59] I. Horrocks, P. F Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, May 2004.

[60] IHS. *FTA-Pro*. http://www.dyadem.com.

[61] International Electrotechnical Commission. *IEC-60812 Analysis techniques for system reliability - Procedure for Failure Mode and Effects Analysis (FMEA)*, 1985.

[62] Isograph Inc. *FaultTree+*. http://www.faulttree.org.

[63] ITEM Software. *ITEM ToolKit*. http://www.itemsoft.com.

[64] P. Jordan. IEC 62304 International Standard Edition 1.0 Medical device software - Software life cycle processes. *The Institution of Engineering and Technology Seminar on Software for Medical Devices*, 2006.

[65] Karlsruhe Institute of Technology (KIT), FZI Research Center for Information Technology, and University of Paderborn. *Palladio: A software architecture simulation approach.* http://www.palladio-simulator.com.

[66] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. *Proc. of the IEEE*, 91:145–164, January 2003.

[67] G. Kiczales, J. Lamping, A. Mehdhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pages 53–60. Springer-Verlag, 1997.

[68] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the Association for Computing Machinery*, 42:741–843, 1995.

[69] G. Klyne and J. J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax.* http://www.w3.org/TR/rdf-concepts, February 2004.

[70] Y. Koji, Y. Kitamura, and R. Mizoguchi. Ontology-based transformation from an extended functional model to FMEA. In *In Proc. of ICED 05*, 2005.

[71] H. Koziolek and J. Happe. A QoS Driven Development Process Model for Component-Based Software Systems. In *Proc. 9th Int. Symposium on Component-Based Software Engineering (CBSE'06)*, volume 4063, pages 336–343. Springer-Verlag, 2006.

[72] B. H. Lee. Using FMEA models and ontologies to build diagnostic models. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 15:281–293, September 2001.

[73] N. Leveson. *Safeware: system safety and computers.* Addison-Wesley, 1995.

[74] D. Lime and O. H. Roux. Formal verification of real-time systems with preemptive scheduling. *Real-Time Syst.*, 41(2):118–151, 2009.

[75] R. R. Lutz and R. M. Woodhouse. Requirements analysis using forward and backward search. *Annals of Software Engineering*, pages 459–475, 1997.

[76] M. Marzolla and S. Balsamo. UML-PSI: The UML Performance SImulator. In *Proc. of the* 1^{st} *Int. Conf. on the Quantitative Evaluation of Systems*, pages 340–341, Enschede, The Netherlands, September 2004. IEEE Computer Society.

[77] D. L. McGuinness and F. van Harmelen. *OWL 2 Web Ontology Language*. http://www.w3.org/TR/owl-features, February 2004.

[78] P. Merlin and D.J. Farber. Recoverability of Communication Protocols. *IEEE Trans. on Comm.*, 24(9):1036–1043, 1976.

[79] J. Merseguer, J. Campos, S. Bernardi, and S. Donatelli. A Compositional Semantics for UML State Machines Aimed at Performance Evaluation. In *Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)*, pages 295–302, Washington, DC, USA, 2002. IEEE Computer Society.

[80] R. J. Mikulak, R. McDermott, and M. Beauregard. *The Basics of FMEA*. Productivity Press, 2008.

[81] K. Mokos, G. Meditskos, P. Katsaros, N. Bassiliades, and V. Vasiliades. Ontology-based model driven engineering for safety verification. In *36th EUROMICRO Conference on SW. Eng. and Advanced Applications, SEAA 2010*, pages 47–54, 2010.

[82] National Aeronautics and Space Administration. *NASA Software Safety Guidebook NASA-GB-8719.13 - NASA TECHNICAL STANDARD*, March 2004.

[83] Object Management Group. *UML Profile for Schedulability, Performance and Time Specification*, January 2005.

[84] Object Management Group. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems v1.0*, 2009.

[85] Object Management Group. *Unified Modeling Language: Infrastructure, Version 2.3*, September 2009.

[86] Object Management Group. *Unified Modeling Language: Superstructure, Version 2.3*, September 2009.

[87] H. Pentti and H. Atte. *Failure Mode and Effects Analysis of software-based automation systems - STUK-YTO-TR 190*. VTT Industrial Systems - STUK, August 2002.

[88] D. Petriu, C. Shousha, and A. Jalnapurkar. Architecture-based performance analysis applied to a telecommunication system. *IEEE Trans. on SW Eng.*, 26:1049–1065, November 2000.

[89] D. B. Petriu and M. Woodside. An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *SW Systems and Modeling*, 6:163–184, 2007.

[90] E. Prud'hommeaux and A. Seaborne. *SPARQL Query Language for RDF*. http://www.w3.org/TR/rdf-sparql-query, January 2008.

[91] PTC Product Development Company. *Windchill Fault Tree Analysis software*. http://www.ptc.com/product/relex/fault-tree.

[92] PTC Product Development Company. *Windchill FMEA official website*. http://www.ptc.com/product/windchill/fmea.

[93] QA Systems - The Software Quality Company. *Cantata++*. http://www.qa-systems.com/cantata.html.

[94] Radio Technical Commission for Aeronautics. *DO-178B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[95] S. Rai. Evaluating FTRE's for Dependability Measures in Fault Tolerant Systems. *IEEE Trans. on Comput.*, 44(2):275–285, Feb. 1995.

[96] E. S. Raymond. *The New Hacker's Dictionary*. The MIT Press, Cambridge, 1991.

[97] D. J. Reifer. Software Failure Modes and Effects Analysis. *IEEE Transactions on Reliability*, R-28(3):247 –249, aug. 1979.

[98] ReliaSoft. *XFMEA official website*. http://www.reliasoft.com/xfmea.

[99] L. Ridi, J. Torrini, and E. Vicario. Developing a scheduler with difference-bound matrices and the floyd-warshall algorithm. *IEEE Software*, 29:76–83, 2012.

[100] O. H. Roux and D. Lime. Time Petri Nets with inhibitor hyperarcs: formal semantics and state-space computation. 25^{th} *Int. Conf. on Theory and Application of Petri Nets*, 3099:371–390, 2004.

[101] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[102] D. C. Schmidt. Model–Driven Engineering. *IEEE Computer*, pages 1–2, February 2006.

[103] D. C. Schmidt, M. Stal, H. Rohnert, F. Buschmann, and J. Wiley. *Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. John Wiley & Sons, 2000.

[104] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. on Comput.*, 39(9):1175–1185, 1990.

[105] C. U. Smith, C. M. Lladó, V. Cortellessa, A. Di Marco, and L. G. Williams. From UML models to software performance results: an SPE process based on XML interchange formats. In *Proc. of the* 5^{th} *Int. Workshop on SW and Performance*, pages 87–98, 2005.

[106] Society of Automotive Engineers'. *SAE J-1739 Potential Failure Mode and Effects Analysis in Design (Design FMEA) and Potential Failure Mode and Effects Analysis in Manufacturing and assembly Processes (Process FMEA) Reference Manual*, 1994.

[107] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. AspectC++: An Aspect-Oriented Extension to C++. In *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS) Pacific 2002*, pages 53–60, 2002.

[108] D. H. Stamatis. *Failure Mode and Effect Analysis: FMEA from Theory to Execution*. Amer Society for Quality, 2003.

[109] The Mathworks. *Simulink*. www.mathworks.com/products/simulink.

[110] M. Towhidnejad, D.R. Wallace, and Jr. Gallo, A.M. Validation of object oriented software design with fault tree analysis. In *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03)*, pages 209 – 215, Dec. 2003.

[111] United States Department of Defense. MIL-STD-1629A - Procedures for Performing a Failure Mode, Effects and Criticality Analysis. Technical report, US Department of Defense, November 1980.

[112] United States Department of Defense. MIL-STD-498, MILITARY STANDARD FOR SOFTWARE DEVELOPMENT AND DOCUMENTATION. Technical report, USDoD, 1994.

[113] United States Department of Defense. MIL-STD-498, SDD Data Item Description. Technical report, USDoD, 1994.

[114] United States Department of Defense. MIL-STD-498, SRS Data Item Description. Technical report, USDoD, 1994.

[115] United States Department of Defense. MIL-STD-498, SSDD Data Item Description. Technical report, USDoD, 1994.

[116] United States Department of Defense. MIL-STD-498, STD Data Item Description. Technical report, USDoD, 1994.

[117] USC Center for Software Engineering. *UCC: Unified Code Count.* http://sunset.usc.edu/research/CODECOUNT.

[118] E. Vicario. Static Analysis and Dynamic Steering of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. on SW Eng.*, 27(1):728–748, August 2001.

[119] W. E. Vesely and F. F. Goldberg and N. H. Roberts and D. F. Haasl. *Fault Tree Handbook*. U. S. Government Printing Office, 1981.

[120] W3C OWL Working Group. *OWL Web Ontology Language.* http://www.w3.org/TR/owl2-overview, December 2012.

[121] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Statshulat, and P. Stenstroem. Priority Inheritance Protocols: The Worst Case Execution-Time problem: Overview of methods and survey of tools. *ACM Trans. Emb. Comp. Sys.*, 7(3):1–53, 2008.

[122] Wind River. *VxWorks*. www.windriver.com/products/vxworks.

[123] R. Wirth, B. Berthold, A. Krämer, and Peter. Knowledge-Based Support of System Analysis for Failure Mode and Effects Analysis. *Engineering Applications of Artificial Intelligence*, 9:219–229, 1996.

[124] W. E. Wong, S. S. Gokhale, and J. R. Horgan. Quantifying the closeness between program components and features. *J. Syst. Softw.*, 54:87–98, October 2000.

[125] W. E. Wong, J. R. Horgan, S. S. Gokhale, and K. S. Trivedi. Locating program features using execution slices. In *Proceedings of the 1999 IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*, ASSET '99, pages 194–203. IEEE Computer Society, 1999.

[126] M. Woodside, D. Petriu, D. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by Unified Model Analysis (PUMA). In *Proc. of the* 5^{th} *Int. Workshop on SW and Performance*, pages 1–12. ACM, ACM, 7 2005.