Cent. Eur. J. Comp. Sci. • 1(3) • 2011 • 294-328
DOI: 10.2478/s13537-011-0025-x

Central European Journal of **Computer Science**

A Formal Framework for Dependability and Resilience from a Software Engineering Perspective

Research Article

Nicolas Guelfi[1]*

1 *University of Luxembourg, LASSY - Laboratory for Advanced Software Systems,*
Computer Science and Communications Research Unit, Faculty of Sciences, Technology and Communication,
6, rue Richard Coudenhove Kalerghi, L-1359 Luxembourg, LU

Received 7 June 2011; accepted 11 September 2011

Abstract: The goal of this article is to provide a rigorous conceptual framework for defining the concepts of dependability and resilience. Since the seventies, the terms dependability and resilience have been used in nearly all the scientific and technological fields related to Information and Communication systems. The introduction and use of these concepts in all these fields makes it difficult to have a common and precise definition. Having such a definition is nevertheless mandatory for the software and systems engineering research community that create development processes, languages and tools to support the engineering of products that would be required to be dependable or resilient. For this, we introduce an abstract and generic terminology defined mathematically to be used when speaking about dependability and resiliency. We also provide some abstract semantic descriptions to these terminological elements. This formal framework is defined from a software engineering perspective, which means that we define its components such that they are useful for the development or improvement of analysis, architectural design, detailed design, implementation, verification and maintenance phases. To this aim, we provide the necessary elements in accordance with a model driven engineering perspective that enable the definition of a new modelling language for dependable and resilient systems.

Keywords: resilience • model driven engineering • conceptual framework • dependability • software engineering • formalization

1. Introduction

Software engineering [40, 46] aims at providing theories, methods and tools to allow for the production of *I*nformation and *C*ommunication *T*echnological systems (ICT systems). Many adjectives can be used to qualify the production and the system built from different perspectives all of which are related to some so-called quality criteria. Two important areas of software engineering[1] are model driven engineering and dependability. Model driven engineering [3, 38] is a

* *E-mail: nicolas.guelfi@uni.lu*
1 *In this document we interchange the terms software engineering and "information and communication technological (ICT) systems engineering" even though we use ICT systems engineering is more appropriate.*

field aiming at proposing methods and tools supporting the engineering of ICT systems for which models are extensively used. A model, in this sense, is an abstraction of a real world entity that is of interest for the engineering of the ICT system under consideration. Models are defined using basic modelling elements and modelling elements' combination operators. Models are expressions (mainly textual or graphical) that comply with a modelling language (defined using a textual or graphical syntax). Examples of known modelling languages in software engineering are UML [42], BPMN [44], Statecharts [26] and of course all programming languages among others[2]. The development of modelling languages is a continuous process that aims at tailoring the language to its targeted use.

The second field of interest, dependability, abstractly characterises the trustworthiness of a computing system. Dependability expresses informally the confidence that can be placed on services delivered (see http://www.dependability.org/). A well known informal conceptual framework has been proposed in [6, 7]. Here a taxonomy is proposed in which dependability concepts are organised into three categories: attributes (availability, reliability, safety, confidentiality, integrity, maintainability), threats (faults, errors, failures) and means (fault prevention, fault tolerance, fault removal, fault forecasting).

From an informal perspective means are used at specific points in the ICT systems's life cycle from its creation until its retirement in order to provide the targeted level for each of the attributes. In the last few years a significant amount of research and development has been dedicated to proposing languages, methods and tools to engineer dependable ICT systems. Using either explicit or implicit means that can be defined statically or dynamically in an autonomous or heteronomous manner. It is easy to understand and demonstrate that the dependability of an ICT system is a first class quality attribute. Even though the taxonomy, referred to above, represents a major improvement in the conceptual clarification of the concepts related to dependability, it is not sufficient for two main reasons. Firstly the taxonomy is not precise enough from a scientific point of view and secondly it has not been tailored to be used by the model driven engineering community. In order to gain precision, a first, simple and commonly agreed upon approach is to provide a mathematical definition. To be compatible with the model driven engineering paradigm, this definition should be given using meta-modelling techniques.

Resilience in ICT systems, introduced around the seventies [16] has been most intensively used within the research community in the very few last years[3]. By reviewing the important references we notice that the word resilience, is used with a variety of definitions and at different levels [9, 39, 53]. With the same intention described previously, it is in the interest of ICT systems engineering community to derive a precise definition of resilience that can be also integrated into modelling languages.

With respect to the mathematical definition of concepts that are useful to understand dependability and resilience, we could use many fields (sets theory, mathematical logic, category theory, mathematical statistics and geometry among others) at different abstraction levels. Field selection should depend on the targeted exploitation of the formalisation. In this article, it is proposed to use algebra (mainly elementary algebra and basis of general algebra focusing on set theory and functions) since it is one of the mostly used mathematical fields by ICT systems engineers and scientists. Its concepts are easily mapped to the ones that exist in technological fields like programming languages and data bases. In addition, the concepts correspond to the terminology used in the natural language of document production throughout the ICT systems procurement life cycle (such as requirement document, verification and validation plans). Last but not least, the abstraction level chosen for the framework proposed in this article makes this choice appropriate. The intention is also that the framework should allow for being refined such that more detailed definitions of its concepts using different mathematical structures may be introduced in a consistent way with the framework. This should be considered at the modelling language definition level. As an example, the dependability and resilience of a system might be dependent upon the metrics that can be defined based on probabilities, statistics, logical systems, model checking or test results. Thus these metrics should of course be present in the language used to model the dependable or resilient system.

In order to allow for the exploitation of the framework proposed in this article (called *DREF* for Dependability and Resilience Engineering Framework) in the modelling language definition, some integration techniques must be provided. To this aim and following the model driven engineering (MDE) perspective, we propose to define a meta-model for the

[2] *The intention is neither to provide a complete list nor categorize all the languages used in ICT systems engineering. Nevertheless, more detail is given in the last section of this document.*

[3] *on the approximately 1300 citations using the term resilient or resilience registered at DBLP, 90% appeared after 2000 and 75% in the last five years.*

DREF framework. Further, we present the process for engineering new domain specific modelling languages (DSL) using a model driven engineering approach. In this process, we define how to exploit the *DREF* meta-model when introducing a new DSL. As proposed in this article, this can be done either by extending the existing DSL to integrate dependability and resilience support or upon creating the new DSLs.
In the following discussion, the formal conceptual framework for dependability and resilience is provided along with some basic illustrations. Then an approach for integrating the proposed framework into a modelling language definition is proposed and illustrated through a simple case study. A critical analysis and perspective section is then proposed to help understand the focus of this work. Finally possible research directions to be followed in the future are discussed.

2. Definition of a formal conceptual framework for dependability and resilience

In this section and section 3, we introduce all the concepts needed to define resilience. The main goal of these sections is to provide a precise mathematical set of definitions for all notions necessary to address resiliency. These two sections might be difficult to follow by a non technical reader. Nonetheless, the examples and explanations given should assist the reader in making this content more accessible.

2.1. Entities, Properties and satisfiability

The two first basic sets of the proposed conceptual framework (named *DREF*) are the sets of *entities* and *properties*. Entities are anything that is of interest to be considered. It might be a program, a database, a person, a hardware element, a development process, a requirement document, et cetera. Properties are the basic concepts to be used to characterise entities. It might be an informal requirement, a mathematical property or any entity that aims at being interpreted over entities.

Definition 1.
The basic sets of the *DREF* conceptual framework, defined as disjoint subsets of a given universe U are:

(i) $PROPERTY \in \wp(U)$ the set of all properties.

(ii) $ENTITY \in \wp(U)$ the set of all entities.

Remark 2.
(a) $prop, prop_a, prop_1 \dots prop_n$, (resp. $ent, ent_a, ent_1 \dots ent_n$) will denote an element of $PROPERTY$ (resp. of $ENTITY$).

(b) Subsets of $PROPERTY$ and $ENTITY$ will be denoted by capitalization (e.g. $Prop, Ent$). Indexed notation might also be used.

The fact that a property is satisfied to some degree by an entity is defined as a function as follows:

Definition 3.
Let $Prop$, Ent be sets of properties and entities; Satisfiability, sat, is a function such that:
$sat : Prop \times Ent \longrightarrow \mathbb{R} \cup \{\perp\}$

Remark 4.
(a) We denote by $dom(sat)$ the subset of $Prop \times Ent$ for which sat is defined.

(b) $sat(prop, ent) = \perp$ represents the fact that the satisfiability of $prop$ for ent cannot be determined (in addition to the fact that sat functions are partial functions). This is to allow one to differentiate the case where a satisfiability value is expected but the satisfiability evaluation cannot be determined to the case where the satisfiability function is not expected.

(c) We use $\mathbb{R}$ as co-domain of the satisfiability functions in order to cover all cases that would be necessary.

Example 5.

A simple satisfiability function could consider its co-domain partitioned using two arbitrary values (e.g. 1 and 0) such that:

(i) $sat(prop, ent) = 1$ represents the fact that $prop$ is "exactly satisfied" by ent (or ent "satisfies exactly" $prop$). In this context, 1 represents the nominal satisfiability.

(ii) $sat(prop, ent) = 0$ represents the fact that $prop$ is "exactly unsatisfied" by ent (or ent "unsatisfies exactly" $prop$). In this context, 0 represents a tolerance threshold.

(iii) $sat(prop, ent) > 1$ (resp. ≤ 0) represents the fact that $prop$ is "oversatisfied" (resp. "under satisfied") by ent (or ent "satisfies more (resp. less) than" $prop$).

(iv) in case, $sat(prop, ent) \geq 1$ (resp. ≤ 0) , we can simply say that $prop$ is "satisfied" (resp. unsatisfied) by ent (or ent "satisfies" $prop$). In this context, 1 (resp. 0) represents an acceptance (resp. rejection) threshold.

(v) if $sat(prop, ent) \in [0, 1]$ then we can consider that $prop$ "is partly satisfied and partly unsatisfied" by ent.

For a strict Boolean satisfiability function only two satisfiability values would be accessible (e.g. 0 and 1). Thus only two notions would be used: satisfied or unsatisfied.

We consider that the notions of under and over satisfiability are necessary to represent the frequent informal situation for which an entity has more (resp. less) than the required property. The existence of an order relation between properties could represent the correlation of satisfiability between properties. When this order relation can be defined it could be useful that it is $strongly\ sat-compatible$ as defined below to indicate that a property greater than another must have its satisfiability value always greater than the one of the other for all the entities considered.

Definition 6.

Let sat be a satisfiability function over $Prop$ and Ent and $<_{prop}$ a partial order defined over $Prop$. $<_{prop}$ is said $strongly\ sat-compatible$ *iff* the following property holds:
$\forall\ prop_a, prop_b \in Prop\ (prop_a <_{prop} prop_b) \Rightarrow (\forall\ ent \in Ent\ sat(prop_a, ent) < sat(prop_b, ent)\ or\ sat(prop_a, ent) = sat(prop_b, ent) = \perp)$

As stated in the introduction, the formal framework developed is intended to cover the current use of these concepts from an informal perspective (e.g. natural language words in expression), through the modelling expression in semi-formal notations up until mathematically based notation. Thus for each context, one should associate, at the right level, the concepts of the *DREF* framework to the existing concepts.

Example 7.

Let us consider the context where mathematical logic is used to describe properties. In this case, entities will be logical structures (a set of logical structures is denoted $LogStruct$ as a member of the power of all possible logical structures $\wp(LOGSTRUCT)$). Properties are logical formulae. Let $LSpec$ denotes a set of logical formulae and $LStruct$ a set of logical structures. Let sat be a satisfiability function such that:

$$sat : LStruct \times LSpec \longrightarrow \mathbb{R} \cup \{\perp\}$$

$$s.t.\ sat(lstruct, p) =$$

$$\begin{cases} 1\ if\ lstruct \models p \\ 0\ if\ lstruct \not\models p \\ \perp\ \ else^{4} \end{cases}$$

Let us now define an order relation $<_l$ over $LSpec$ s.t. $p <_l p'$ *iff* $p \vDash p'$. Thus we have the following theorem:

Theorem 8.
$<_l$ is strongly $sat-compatible$.

Proof. $\forall\ lstruct \in LStruct\ sat(lstruct, p) = 1 \Rightarrow lstruct \vDash p$. Furthermore we know that $p <_l p' \Rightarrow p \vDash p'$. So, $lstruct \vDash p'$ and so $sat(lstruct, p') = 1$. □ □

The same approach is possible if the properties would be algebraic specifications with a loose semantics and entities would be algebraic structures with a loose semantics [57].

Table 1. Entities.

1	RF	React Framework

Table 2. Properties.

1	SOA	is SOA oriented
2	SPL	is a SPL framework

Table 3. Observers.

1	NG	Nicolas Guelfi
2	BR	Benoit Ries
3	JL	Jerome Leemans

Example 9.
In this example, we consider an entity that corresponds to a first version of a software product line (SPL) platform developed in our laboratory [34] (see Table 1). This SPL platform (RF for REACT Framework) is used to derive Crisis Management Systems (e.g. car crash crisis, fire crisis in schools, pollution crisis). There are two properties to consider (see Table 2). The first states the compliance of the platform w.r.t. the service oriented architecture style [17] and the second the compliance of the entity RF with the definition of a SPL platform according to [45]. The decision depends on three stakeholders (lets call them observers[5] as given in Table 3. A satisfiability function is given for each observer and for the REACT framework entity in Table 4. Each observer provides its satisfiability function. In this example, each property is evaluated using a discrete evaluation scale of naturals of $[-5, +5]$ (with acceptance and rejection thresholds both set at 0).

2.2. Subjectivity of satisfaction using observers and balancing

Satisfaction is not an objective concept as illustrated in example 9. To represent subjectivity, we introduce the concept of "observer". Thus the satisfiability functions can be associated to observers.

Definition 10.
The set of all observers is defined as a subset of a given universe U and is denoted $OBSERVER$.

[4] *in the case of incomplete theories*
[5] *the notion of observer is formally defined in the formal framework in the following sections*

Table 4. Satifiabillity function.

Satifiabillity function for RF		
Obs.	prop.s	sat.
NG	SOA	1
	SPL	-2
BR	SOA	2
	SPL	1
JL	SOA	4
	SPL	$\perp$

Remark 11.

Let o be an observer, then sat_odenotes a satisfiability function for the observer o.

We now define the satisfiability function for a set of observers. At this point, no information is available in order to differentiate between observers. Of course, useful satisfiability functions need not only to allow for balancing observers but also properties. This is done later by introducing weights. In the meantime, we use the arithmetic average. Nonetheless, it must be reminded that the satisfiability value for each observer is free to be fixed.

Definition 12.

Let Obs be a set of observers, and Sat_{Obs} a Obs-indexed family of satisfiability functions, then the satisfiability function for Obs is defined as:

$$sat_{Obs} : Prop \times Ent \longrightarrow \mathbb{R} \cup \{\perp\}$$

$$s.t.\ Prop = \bigcup_{o \in Obs} Prop_o \ and\ Ent = \bigcup_{o \in Obs} Ent_o \ and$$

$$sat_{Obs}(prop, ent) =$$

$$\begin{cases} \perp \ if \exists o \in Obs / sat_o(prop, ent) = \perp \\ \frac{\sum\limits_{o \in Obs} sat_o(prop,ent)}{|Obs|} \quad otherwise \end{cases}$$

Remark 13.

We consider that satisfiability of a property p or an entity e can only be defined for a set of observers if $< p, e >$ belongs to the domain of each element of Sat_{Obs}.

Observers and properties might be balanced to reflect the fact that the global satisfiability function might be impacted differently by observers or properties. In many situations, the final satisfaction should take into account that observers are not always equal to each others and that some properties can count more than others. We thus introduce a notion of balancing in a generic way to cover, later, observers and properties too. As a first approach we consider weights (i.e. balancing values) as being positive and non-null values. To avoid considering an observer, it should be removed from the list of observers. A negative weight for an observer would mean that we should consider the opposite of all his judgements!

Definition 14.

Let S be a set, then a balancing of S is defined as a function ω_S such that: $\omega_S : S \longrightarrow \mathbb{R}_+^*$

Remark 15.
We will consider an observer's balancing (e.g. ω_{Obs}) and a property's balancing (e.g. ω_{Prop}).

Definition 16.
Let sat be a satisfiability function over $Prop$ and Ent, ω_{Prop} a balancing of $Prop$, then the global balanced satisfiability of sat, is denoted as $gsat_{\omega_{Prop}}$ and is such that:

$$gsat_{\omega_{Prop}} = \frac{\sum\limits_{<p,e>\in dom(sat)} \omega_{Prop}(p) \times sat(p,e)}{\sum\limits_{p\in Prop} \omega_{Prop}(p)}$$

Definition 17.
Let Obs be a set of observers, Sat_{Obs} an Obs-indexed set of satisfiability functions, ω_{Obs} a balancing of Obs. A balanced satisfiability function for Obs, Sat_{Obs} and ω_{Obs} is denoted as $sat_{\omega_{Obs}}$ and is such that:

$$sat_{\omega_{Obs}} : Prop \times Ent \longrightarrow \mathbb{R} \cup \{\bot\}$$

$$s.t.\ Prop = \bigcup_{o\in Obs} Prop_o\ and\ Ent = \bigcup_{o\in Obs} Ent_o\ and$$

$$sat_{\omega_{Obs}}(p,e) =$$

$$\begin{cases} \bot\ if \exists o \in Obs / sat_o(p,e) = \bot \\ \frac{\sum\limits_{\substack{o\in Obs \\ /<p,e>\in dom(sat_o)}} \omega_{Obs}(o) \times sat_o(p,e)}{\sum\limits_{o\in Obs} \omega_{Obs}(o)} & otherwise \end{cases}$$

Since observers can be grouped we must analyze the properties of the satisfiability functions. We thus define the notion of coherence.

Definition 18.
Let Sat_{Obs} be a Obs-indexed set of satisfiability functions. Sat_{Obs} is said to be coherent *iff*

$$(\forall o,o' \in Obs)(\forall p \in PROPERTY)(\forall e \in ENTITY)$$
$$\neg(sat_o(p,e) = \bot \wedge sat_{o'}(p,e) \neq \bot)$$
$$\wedge \neg(sat_o(p,e) < 0 \wedge sat_{o'}(p,e) > 0)$$

Definition 19.
Let Sat_{Obs} be a Obs-indexed set of satisfiability functions. Sat_{Obs} is said to be homogeneous *iff* coherent and

$$(\forall o,o' \in Obs)(\forall p \in PROPERTY)(\forall e \in ENTITY)$$
$$(sat_o(p,e) \leq 0 \wedge sat_{o'}(p,e) \leq 0)$$
$$\vee(sat_o(p,e) \geq 0 \wedge sat_{o'}(p,e) \geq 0)$$

Example 20.
If we consider the same case described in example 9, then we can define weights for observers and properties as given in Table 5. We can see that the observer *NG* (considered head of the project) has a weight of 3 compared to the weight 1 of *JL* (a master trainee). Concerning properties, all the observers share the same property weights which indicate that the software product line dimension (*SPL*) of the framework developed (*RF*) is twice more important that the software service oriented one (*SOA*).
Thus we have the following global balanced satisfiability values for RF:

- for the observer NG it is $gsat_{NG} = -1$.
- for the observer BR it is $gsat_{BR} = 4/3$.
- for the observer JL it is $gsat_{JL} = 4$.
- for the set of observers $Obs = \{NG, BR, JL\}$, we have a global balanced satisfiability of $\frac{3\times gsat_{NG}+2\times gsat_{BR}+1\times gsat_{JL}}{3+2+1} = \frac{11}{18} \approx 0.61$.

  Of course, since JL is not capable of evaluating the SPL property over the RF entity, the family of functions is not domain-homogeneous. Thus the computation of the global satisfiability is biased. If we compute the maximum set of domain-homogeneous observers from Obs (i.e. $Obs_h = \{NG, BR\}$), then the global satisfiability of Obs_h is $\frac{-1}{15} \approx -0.06$ and thus we move from a positive (above the acceptance threshold) to a negative value (i.e. below the rejection threshold). This result indicates that the scale chosen[6] has changed from an acceptance status to a rejection status.

Table 5. Weights and Satisfiabillity function.

Weights and Sat for RF				
Obs.	w.o.	prop.	w.p.	sat.
NG	3	SOA	1	1
		SPL	2	-2
BR	2	SOA	1	2
		SPL	2	1
JL	1	SOA	1	4
		SPL	2	⊥

It must be noticed that the global satisfiability function is not intended to be the only or the primary satisfiability information to be used when exploiting the *DREF* framework. Doing so would imply the exclusion of all approaches that would be required to handle specific properties or observers since they would be hidden in the global weighted average computation.

Balancing is an important concept for explicit definition of priorities. In all the engineering projects we have been involved in (from 10k€ to 3 M€) implicit or explicit prioritization, ordering, or balancing of properties were incorporated by partners (i.d. observers), themselves implicitly or explicitly weighted. Nevertheless, it is true that the current practices have difficulties in explicitly stating those weights. This is also the case for explicit modeling of some types of faults, especially the ones that are of all the following types: development, internal, human made, software level, non malicious, non deliberate, incompetence due and persistent. We believe, nevertheless, that accurate scientific models should be able to handle them. Further work is required in experimenting with methodologies in which the weights can be efficiently incorporated.

2.3. Change, Evolution Axis and Correlations

The terminology used in the fields related to Information and Communication Technological (ICT) systems, quite frequently incorporates the following keywords: change, evolution, adaptation, variation, modification, transformation. If we focus on a program as an entity, then a program change could refer to a new version of the program seen as a sequence of lines of code, or it might refer to the change of some program "status" defined using specific "state variables". One can easily see that those two interpretations of program change are fundamentally different.

[6] *see example 9.*

Considering the notion of change seems logical since we are currently considering ICT systems and humans as being entities whose existence (i.e. definition) seems to change with time (at least). A first simple approximation, then, would be to define change as the difference between two definitions of two entities distributed over a common evolution axis. As an example, let consider p_1 as a simple imperative program sorting a list of integers using the bubble algorithm and p_2 using a quick sort algorithm. If we consider p_1 and p_2 as comparable entities then the change from p_1 to p_2 should represent the difference between the two programs. The way of defining the difference is thus fundamental. A simple very informal definition could be: "p_1 differs from p_2 by the type of sorting algorithm used". A more precise definition could be provided by the use of a term rewriting function $rewrite$ that would rewrite a bubble sort imperative program on to a quick sort based imperative program. In this case $p_2 = rewrite(p_1)$, will be used to define the difference between the two programs in terms of all the modifications made to p_1 to reach p_2.
In case of a program status change, a program change would be defined as any modification to any of the predefined state variables constituting the program status. If the program status of p_1 is defined by the status of a local variable containing the list of integers to be sorted then p_1 and p_2 could not be considered as equivalent at any point in the evolution axis except at initial and final evolution points. In this last case, the difference between the two programs is expressed as a difference between two lists of integers. Thus the definition of the evolution axis is a mandatory preliminary step to allow an entity's comparison.

Definition 21.
An evolution axis is a set of values that are used to index a set of entities or a set of properties.

Remark 22.
The intention is to allow for comparison of entities relative to an evolution axis. Concerning ICT systems, the commonly used axes are the time axis (that can be considered as discrete or continuous) related to system's versioning or related to system status. If we consider software product lines then one evolution axis can be related to variants.

Example 23.
If we consider the REACT framework (entity RF), we have three versions of the framework. While the first is that described in 9, the second has an added service discovery mechanism based on a service registry, and the third has provided service orchestration of reusable modules corresponding to different types of crisis management scenarios belonging to different crisis types explicitly provided in the framework. We then introduce an evolution axis representing the three successive versions of the REACT framework. The evolution axis is then the set $\{v_1, v_2, v_3\}$ and it concerns the entity RF. We will then have three values on this evolution axis: $RF_{v1}, RF_{v2}, RF_{v3}$. It is important to notice that those values are strictly ordered. This is mandatory to consider when addressing resilience.

Example 24.
Another more complex illustration can be made in the context of the relationships between requirements and implementation. A classical situation in a system's life cycle is the fact that the correspondence between requirements and realisations is not always optimal[7]. Of course the problem is to define what the criteria are to evaluate and order these correspondences. A simple approach could be to compare the set of user functionalities described in the requirement document and the set of functionalities supported by the system. This corresponds to a comparison between the requirement document obtained at the analysis level and a reverse engineering of the requirement from the implemented system. In this case, one can define two evolution axes, one for the implemented system and one for the requirements. We thus have the properties' evolution axis $Prop_{evo_1} = \{v_1, v_2, v_3, v_4\}$, and the system's implementation evolution axis $Ent_{evo_1} = \{v_1, v_2, v_3, v_4\}$. We have four values for the properties $Prop = \{req_{v1}, req_{v2}, req_{v3}, req_{v4}\}$ and three values for the system entities $Ent = \{sys_{v1}, sys_{v2}, sys_{v3}\}$. The satisfiability function would then evaluate the adequacy of

[7] *We have presented this problem in [5] and [4] where the proposed solution was to use the system's abstract architecture layer to evolve the system and to reduce the gap between the requirements and implementation.*

requirements vs implementations. This function could be defined as a percentage of the functionalities of requirements covered by the system[8]. In this case, we could have the satisfiability functions defined in the tables 6,7 and 8. Depending on the evolution processes definition and coordination for the requirements and for the implementations, the set of adequacies to be evaluated is defined characterising the domain of the satisfiability function.

Table 6. Weights and Satisfiability function for sys-v1.

Weights and Sat for sys-v1				
Obs.	w.o.	prop.	w.p.	sat.
NG	1	req-v1	1	75
		req-v2	1	60

Table 7. Weights and Satisfiability function for sys-v2.

Weights and Sat for sys-v2				
Obs.	w.o.	prop.	w.p.	sat.
NG	1	req-v2	1	80

Table 8. Weights and Satisfiability function for sys-v3.

Weights and Sat for sys-v3				
Obs.	w.o.	prop.	w.p.	sat.
NG	1	req-v3	1	85
		req-v4	1	90

2.4. Nominal Satisfiability and Requirements

In order to formalise the notion of quality, we must introduce the concept of a nominal satisfiability function and the concept of requirement.

Definition 25.
A nominal satisfiability function is a satisfiability function used to represent the expected satisfiability and to allow comparative evaluation w.r.t. to any satisfiability function that is provided.

Definition 26.
A requirement for a set of entities, Ent, is a set of properties, Req of $PROPERTY$, together with a nominal satisfiability function, $nsat$, such that $dom(nsat) =< Req, Ent >$

Remark 27.
When not provided, the default nominal satisfiability function is such that $\forall < p, e > \in dom(nsat) \quad nsat(p, e) = 1$.

[8] *Of course more precise and rigorous adequacy could be defined. For example, we could here define a satisfiability function where each functionality would be associated with a property and the entity would be the system.*

Definition 28.

Let $rqt =< Req, nsat >$ be a requirement for a set of entities, Ent, and sat a satisfiability function. sat is said to satisfy rqt *iff* $\forall < r, e > \in dom(sat) \quad sat(r, e) \geq nsat(r, e)$.

Example 29.

If we consider the REACT framework (entity RF), we have three versions of the framework as described in example 23. The tables 9, 10 and 11 given below[9] describe the values of the satisfiability functions for the three observers and the same properties.
In this example, we have a nominal satisfiability function $nsat(r, e) = 1$ for all versions of the RF entity and for the two properties. The evolution of the global satisfiability function is the following:
$gsat(`RF - v1`) \approx 0.61$
$gsat(`RF - v2`) \approx 1.22$
$gsat(`RF - v3`) = 3$
The evolution of the global balanced satisfiability function is the following:
$gsat_{w_{Obs}}(`RF - v1`) \approx -0.06$ (in case of a selection of homogeneous observer set)
$gsat_{w_{Obs}}(`RF - v2`) \approx 0.86$
$gsat_{w_{Obs}}(`RF - v3`) \approx 2.93$

This means that $RF - v1$ and $RF - v2$ satisfy the requirements if we do not take into account the balancing ($gsat$ is greater than the nominal satisfiability defined at 1) and that only $RF - v3$ satisfies the requirements if we consider balancing (the two others $-0, 06$ and 0.86 are lower than 1).

Table 9. Weights and Satisfiability function for RF-v1.

Weights and Sat for RF-v1				
Obs.	w.o.	prop.	w.p.	sat.
NG	3	SOA	1	1
		SPL	2	-2
BR	2	SOA	1	2
		SPL	2	1
JL	1	SOA	1	4
		SPL	2	nil

Table 10. Weights and Satisfiability function for RF-v2.

Weights and Sat for RF-v2				
Obs.	w.o.	prop.	w.p.	sat.
NG	3	SOA	1	3
		SPL	2	-1
BR	2	SOA	1	3
		SPL	2	1
JL	1	SOA	1	5
		SPL	2	2

Figure 1 gives a general graphical representation of the satisfiability functions (Y axis) for all the observers (colors) over the properties (Z axis) and the REACT Framework entity evolutions (X axis). For example, we can see the progression of the satisfiability for NG for each property and along the evolutions of RF.

[9] *w.o (resp. w.p.) is the weight for an observer (resp. a property).*

Table 11. Weights and Satisfiability function for RF-v3.

Weights and Sat for RF-v3				
Obs.	w.o.	prop.	w.p.	sat.
NG	3	SOA	1	4
		SPL	2	1
BR	2	SOA	1	3
		SPL	2	5
JL	1	SOA	1	4
		SPL	2	3

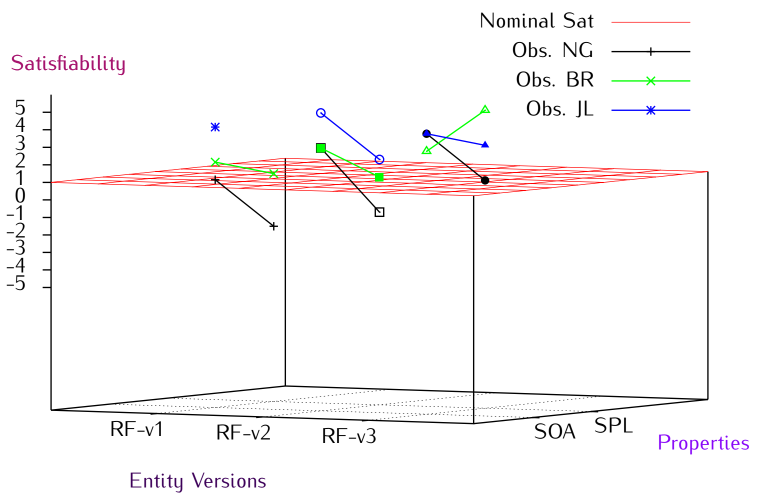


Figure 1. RF Evolution Satisfiability Function represented in 3D.

2.5. Tolerance, Preservation, Improvement and Degradation

In order to address the main concepts of dependability, we propose to define four basic concepts in our formal framework: tolerance, preservation, improvement and degradation. We consider that there is no universal notion of quality but rather that quality is a subjective notion relative to a set of properties that represent the expectation on a specific entity.

Definition 30.

A tolerance threshold is defined as a satisfiability function. Tolerance threshold functions are used to represent the lower bound that defines the tolerance margin for satisfiability functions w.r.t. a nominal satisfiability function.

Remark 31.

Tolerance threshold functions will be denoted as $tolsat$.

The tolerance margin is the space between $nsat$ and $tolsat$ (see Figure 2).

Intolerance is characterised by $nsat = tolsat$.

Definition 32.

A preservation is defined as constancy in a satisfiability function w.r.t. an evolution axis.

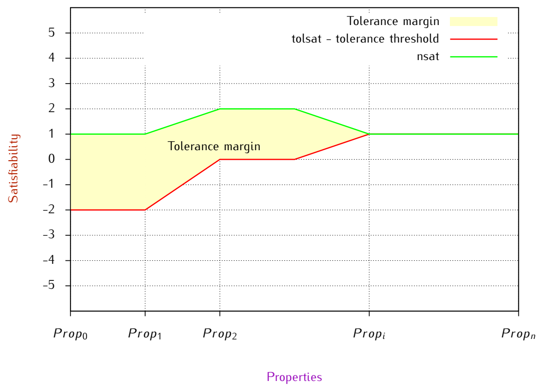


Figure 2. Tolerance threshold.

Definition 33.

An improvement is defined as an increase in a satisfiability function w.r.t. an evolution axis.

Definition 34.

A degradation is defined as a decrease in a satisfiability function w.r.t. an evolution axis.

Remark 35.

Preservation (resp.improvement, degradation) might be observed relative to a nominal satisfiability and tolerance threshold in order to discriminate between different types of preservation (resp.improvement, degradation). As an example, an improvement causing a satisfiability function to go from below to above the tolerance threshold would be characterised as a failure reducing improvement.

Example 36.

We consider the REACT framework and the three versions of the framework described in example 23 for which the satisfiability function is drawn in 2D in Figure 3. Considering the nominal satisfiability function $nsat$, we notice that for almost all pairs of observers (NG,BR,JL) and properties (SOA, SPL), we have continuous improvement along the entity's evolution axis representing the React Framework versions ($RF_{v_1}, RF_{v_2}, RF_{v_3}$). For $< BR, SOA >$, we have an improvement from RF_{v_1} to RF_{v_2} and a preservation from RF_{v_2} to RF_{v_3} . For $< JL, SOA >$, we have an improvement from RF_{v_1} to RF_{v_2} and a degradation from RF_{v_2} to RF_{v_3}. If we now consider the varying nominal satisfiability function, $nsat'$, we observe that there is no improvement but preservation for $< BR, SOA >$ (and $< NG, SPL >, < JL, SOA >$) RF_{v_1} to RF_{v_2}.

2.6. Tolerance and Failure

Based on the previous definition of tolerance margin, we now address the notions of tolerance and failure. Both concepts are related to the satisfaction of a property over an entity given a nominal satisfiability and tolerance threshold functions.

Definition 37.

Given sat, a satisfiability function, and $tolsat$, a tolerance threshold, a failure is defined as a tuple $< r, ent > \in dom(sat) \cap dom(tolsat)$ such that: $sat(r, ent) \leq tolsat(r, ent)$.

Remark 38.

1. We will write $fail(r, ent)$ to denote the failure $< r, ent >$.

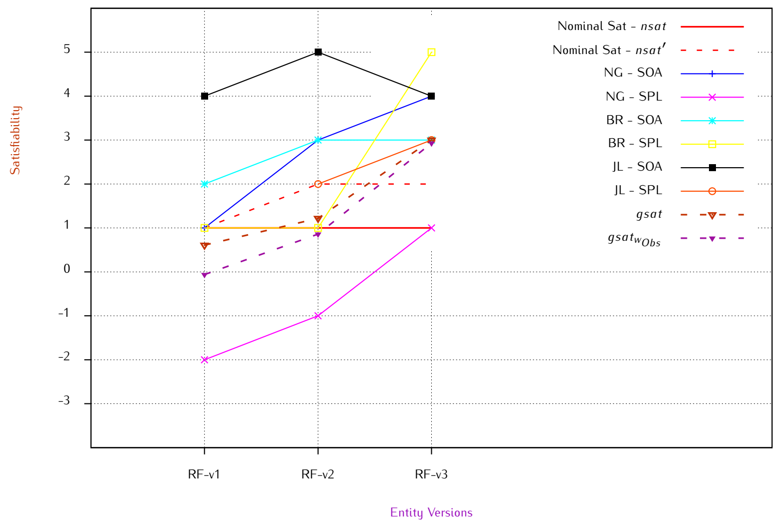


Figure 3. RF Evolution Satisfiability Function 2DGraph.

2. The fact that $\exists < r, ent > \in dom(sat) / sat(r, ent) < nsat(r, ent)$ is considered as a degradation of requirement satisfaction until $sat(r, ent) \leq tolsat(r, ent)$, a situation in which we have a failure.

3. An infinite degradation is a tuple $< r, ent >$ that is a failure for which there is no entity ent' onward in the evolution axis of ent such that $sat(r, ent') > tolsat(r, ent)$.

4. If we want to represent degradation modes it is sufficient to provide a strictly ordered family of tolerance thresholds that partition the tolerance margin into tolerance spaces, that correspond to the different degradation modes.

Definition 39.

Let sat be a satisfiability function, $nsat$ a nominal satisfiability function and $tolsat$ a tolerance threshold. Then a tolerance is defined as a tuple $< r, ent >$ such that:
$< r, ent > \in dom(sat) \wedge < r, ent > \in dom(tolsat) \wedge < r, ent > \in dom(nsat)$
$\wedge\ sat(r, ent) < nsat(r, ent) \wedge sat(r, ent) > tolsat(r, ent)$.

Thus in order to allow the characterisation of fault-tolerance at the requirement level, we need to introduce the notion of a requirement with fault-tolerance[10].

Definition 40.

Let $< Req, nsat >$ be a requirement and $tolsat$ a tolerance threshold, then a requirement with fault-tolerance Req_{FT} is defined as a tuple $< Req, nsat, tolsat >$.

As an example, let us consider a set of comparable[11] entities, $p_{t_1}, \dots, p_{t_n}$, as being n number of changes of a program, p, over the $time$ evolution axis. Let $sat, nsat, tolsat$ be satisfiability, nominal satisfiability and tolerance threshold functions whose values are sketched in Figure 4. We have two failures between P_{t_j} and $P_{t_{j+r}}$ which are the situations where sat (in black) falls below the $tolsat$ tolerance threshold (in red). From $P_{t_{j+r}}$ till P_{t_k} sas is within the tolerance

[10] *Using imprecise language one might infer a requirement with fault tolerance using the terminology 'a fault tolerant requirement'. In our framework, this later expression would have a different semantics since it would consider the requirement as being the entity of interest for which we would have to define a tolerance margin.*
[11] *see remark 22 of evolution axis definition.*

limits characterised by having sas between $nomsat$ (in green) and $tolsat$. Of course we must precise the context of the $sat, nsat, tolsat$ functions which will mainly be dependent on the properties and observers and which should be coherent for the three functions in order to allow a precise interpretation of the tolerance, failure and preservation situations. Positive (resp. negative) variations of sat correspond to improvement (or degradation) w.r.t. satisfiability and constancy (from P_{t_0} to P_{t_1} and from P_{t_l} onward) corresponds to preservation.

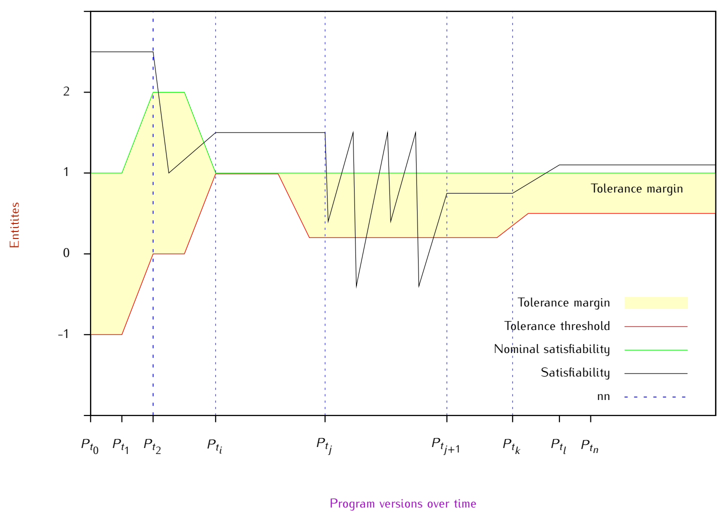


Figure 4. Tolerance threshold.

3. Resilience as change for improvement

We define the general concept of resilience intuitively as a property of an evolution process that is considered to improve capabilities thus avoiding failures and reducing degradations. Roughly, it is the existence of a change toward improvement that reduces failures and tolerance needs. This fact implies that at least one evolution axis exists for resilience definition. Another evolution axis might be introduced as a refinement. The consideration of additional evolution axes can be useful to introduce different evolution types and to study their correlation and impact on resiliency. Different types of resilience might be introduced that depend on the evaluation of the expected reduction of failures and tolerances that are induced by the evolution. The properties used for the evaluation of the satisfiability function define observation points. Thus an observation axis could also be introduced to classify the observation points used for the evaluation of failures and degradations and their change over the chosen evolution axes. The explicit conceptualisation of evolutions and observations is a fundamental task to allow for good dependability and resilience evaluation.

The next definitions are fundamental in our framework. The first introduces: $tolmax$ as the maximum possible level of tolerance needed. It is defined based on the differences between the expected lowest acceptable satisfiability levels (in Figure 6 it is the yellow surface); $stol$ represents the total quantity of tolerance deduced from the effective satisfiability all along the evolution axis (in Figure 6 it is the green surface); $ltol$ is the proportion of the two previous quantities. The second definition introduces a variation of the required tolerance levels between two evolutions (successive or not). The third definition focuses on failures and introduces: 1) a failure level that indicates the failure grade for a given property and entity evolution (in Figure 6 it is the distance between the red curve and any point of an entity version curve that goes below it); 2) $qfail$ represents the number of times an entity is failing (goes below the tolerance threshold) all along its evolution axis; and 3) $sfail$ quantifies the level of failure as for $stol$. The fourth definition introduces the variation quantities in terms of number of failures ($\Delta qfail$) and failure level ($\Delta fail$).

Definition 41.
Let ent_v be an entity, and $ev = \{1, ..., m\}$ be an evolution axis for ent_v. Let sat be a satisfiability function defined over ent_v and over a set of properties $Prop$, $nsat$ a nominal satisfiability function and $tolsat$ a tolerance threshold defined over ent_v along the evolution axis. The notions of maximum tolerance ($tolmax^v$), cumulative tolerance ($stol^v$) and tolerance level ($ltol^v$) are defined as follows:

$$tolmax^v = \sum_{\substack{t \in ev \\ p \in Prop}} (nsat(p, ent_v^t) - tolsat(p, ent_v^t))$$

$$stol^v = \sum_{\substack{t \in ev \\ p \in Prop}} \left(\begin{smallmatrix} Max(sat(p,ent_v^t),nsat(p,ent_v^t)) \\ -Max(sat(p,ent_v^t),tolsat(p,ent_v^t)) \end{smallmatrix} \right) \quad (0 \; if \; Prop = \emptyset)$$

$$ltol^v = \frac{stol^v}{tolmax^v}$$

Definition 42.
Let ent_i *and* ent_j be two entities both evolving along an evolution axis $ev = \{1, ..., m\}$. Let sat be a satisfiability function defined over the cited entities and over a set of properties $Prop$, $nsat$ a nominal satisfiability function and $tolsat$ a tolerance threshold defined over the entities along the two evolution axes. The notion of tolerance variation ($\Delta tol_{i,j}$) induced by the evolution from ent_i to ent_j is defined as follows:

$$\Delta tol_{i,j} = ltol^i - ltol^j$$

Definition 43.
Let ent_v be an entity, let $ev = \{1, ..., m\}$ be an evolution axis for the entity ent_v. Let sat be a satisfiability function defined over ent_v and over a set of properties $Prop$, $nsat$ a nominal satisfiability function and $tolsat$ a tolerance threshold defined over ent_i along the evolution axis. The notions of local failure ($fail^v$), cumulative failure quantity ($qfail^v$) and cumulative failure level ($sfail^v$) are defined as follows:

$$fail^v : ev \times Prop \longrightarrow \mathbb{R} \; / \; fail^v(t, p) = \left(\begin{smallmatrix} tolsat(p,ent_v^t) \\ -Min(sat(p,ent_v^t),tolsat(p,ent_v^t)) \end{smallmatrix} \right)$$

$$qfail^v = |\{< t, p > / fail^v(t, p) > 0\}|$$

$$sfail^v = \sum_{\substack{t \in ev \\ p \in Prop}} fail^v(t, p) \quad (0 \; if \; Prop = \emptyset)$$

Definition 44.
Let ent_i *and* ent_j be two entities attached[12] to a common evolution axis $ev = \{1, ..., m\}$. Let sat be a satisfiability function defined over the cited entities and over a set of properties $Prop$, $nsat$ a nominal satisfiability function and $tolsat$ a tolerance threshold defined over the entities along the two evolution axes. The notions of failure level variation ($\Delta fail_{i,j}$) and failure quantity variation ($\Delta qfail_{i,j}$) induced by the evolution from ent_i to ent_j is defined as follows:

$$\Delta fail_{i,j} = sfail^i - sfail^j$$

$$\Delta qfail_{i,j} = qfail^i - qfail^j$$

[12] *by default* $i \leq j$

The next definition then, provides a first basic definition of resilience as a property over two entities belonging to a common evolution axis. $resil^{T}$ *iff* the tolerance level has decreased; $resil^{F}$ *iff* the number of failures as decreased; and $resil^{TF}$ if both previous properties are true.

Definition 45.

Let ent be an entity evolving along the evolution axis $ev = \{1, ..., n\}$ (the generative axis). Let sat be a satisfiability function defined over the cited entities and over a set of properties $Prop$, $nsat$ a nominal satisfiability function and $tolsat$ a tolerance threshold defined over the entities along the evolution axis. Let $i, j \in ev$, the properties of **T-resilience**,**F-resilience** and **TF-resilience** (noted $resil^{T}_{i,j}$, $resil^{F}_{i,j}$, $resil^{TF}_{i,j}$) are defined as follows:

$$\begin{array}{ll} (i) & resil^{T}_{i,j} \text{ is true iff } \Delta tol_{i,j} > 0 \\ (ii) & resil^{F}_{i,j} \text{ is true iff } \Delta qfail_{i,j} > 0 \\ (iii) & resil^{TF}_{i,j} \text{ is true iff } resil^{T}_{i,j} \wedge resil^{F}_{i,j} \end{array}$$

The definition of TF-resilience is a strict definition of resilience since it requires that both the level of tolerance and the number of failures if any are reduced.

Remark 46.

In case we want to address property (resp. properties set) specific resilience, we must define the set Prop used to compute the different types of resilience. By default, we will consider $Prop$ as the set of properties over which sat is defined.

To illustrate these definitions, let us consider $ev_1 = versions = \{1, ..., n\}$ be the n evolutions steps of an entity ent over the versioning axis and $obs = times = \{k_0, ..., k_m\}$ the $m + 1$ observation points for ent corresponding to a time axis. ent^{k}_{i} represents the entity at version i and time k. In Figure 5, we provide the graph of the satisfiability functions for ent_i, ent_{i+1} *and* ent_{i+2} for the time observation points from k_0 to k_{10}. We consider only one property, a constant $tolsat$ function of 0 and a constant $nsat$ function of 1. If we compute $\Delta fail$ *and* Δtol for the three evolutions ent_i, ent_{i+1} *and* ent_{i+2}, we obtain the figures given in Table 12 and in Table 13. We notice that the tolerance level $ltol$ decreases from 0.86 to 0.23, which corresponds to a Δtol of 0.63. From a graphical point of view, we observe that through the two evolution steps, the total level of tolerance is constantly reduced (this is represented approximately by the surface reduction (i.e. from the surface in between sat_{ent_i} and $nsat$; and the surface in between $sat_{ent_{i+1}}$ and $nsat$). Thus we can easily prove that there is a resilient process over ev_1 in between i and $i+2$ (i.e. $resil^{TF}_{i,i+1} \wedge resil^{TF}_{i+1,i+2}$ is true).

Table 12. Values for $tolmax, tol, ltol$.

Entity	$tolmax$	tol	$ltol$
ent_i	11	9,5	0.86
ent_{i+1}	11	7	0.64
ent_{i+2}	11	2.5	0.23

Table 13. Values for Δtol and $\Delta fail$.

$Entities\ couple$	Δtol	$\Delta fail$	$\Delta qfail$
ent_i, ent_{i+1}	0,23	1.5	2
ent_{i+1}, ent_{i+2}	0,41	1.5	1
ent_i, ent_{i+2}	0,64	0	3

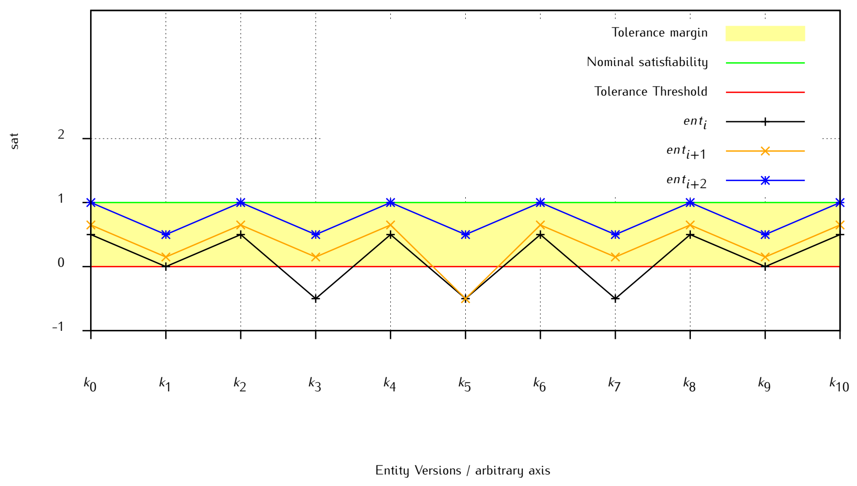


Figure 5. Simple Satisfiability correlation arbitrary axis / versioning axis.

In the general case, $nsat$ and $tolsat$ may vary and $sat_{ent_{i+1}}$ is not a constant improvement over the two evolution axes w.r.t. sat_{ent_i}. Figure 6 demonstrates such a case[13]. From a surface point of view, we are interested in observing the tolerance surface (in between $tolsat$ and $nsat$), which is between the satisfiability gap through the evolution axis (i.e. the surface in between sat_{ent_i} and $sat_{ent_{i+1}}$).
We can draw the following conclusions concerning Figure 6:

- (+) represents the tolerance that is removed by evolving from ent_i to ent_{i+1}. ent_{i+1} is thus considered as an improvement w.r.t. ent_i for the associated evolution steps over $time$.
- (-) is the tolerance that is added by evolving from ent_i to ent_{i+1}. It is a result of the fact that ent_{i+1} represents a degradation w.r.t. ent_i for the corresponding evolutions over $time$ axis.
- (i) corresponds to failures not suppressed by the evolution.
- (ii) is a situation where the evolution has made a satisfactory entity evolve into a failing entity.
- (1) and (2) represent no improvement w.r.t. tolerance or to failure has been accomplished by the evolution. Both entity evolutions are over or below the tolerance margin.

If we compute $\Delta fail\ and\ \Delta tol$ for the two entities for all the evolutions from k_0 to k_{10}, we notice that the tolerance level $ltol$ decreases from 0.43 to 0.12 corresponding to Δtol of 0.21. The total level of tolerance is reduced by a factor 3.5. Thus we can deduce that as soon as Δtol is greater (resp. lower) than 0 we have an improvement (degradation) of tolerance needs due to the evolution. The number of failures, is constant at 3. Thus according to our definition there is no resiliency because we did not reduce the number of failures.
Resilience can now be analysed by observing its values over the chosen evolution axis. The current notion of resilience is given in definition 45. It considers global tolerance and the total number of failures. We are free to define specific notions of resilience depending on a number of interesting parameters. A non-restrictive list could be:

- minimum, maximum delays in term of evolution steps needed to change Δtol and $\Delta qfail$.
- use of $\Delta qfail$ instead of $\Delta fail$.

[13] *Still considering a fixed property or considering a global balanced satisfiability function over the properties.*

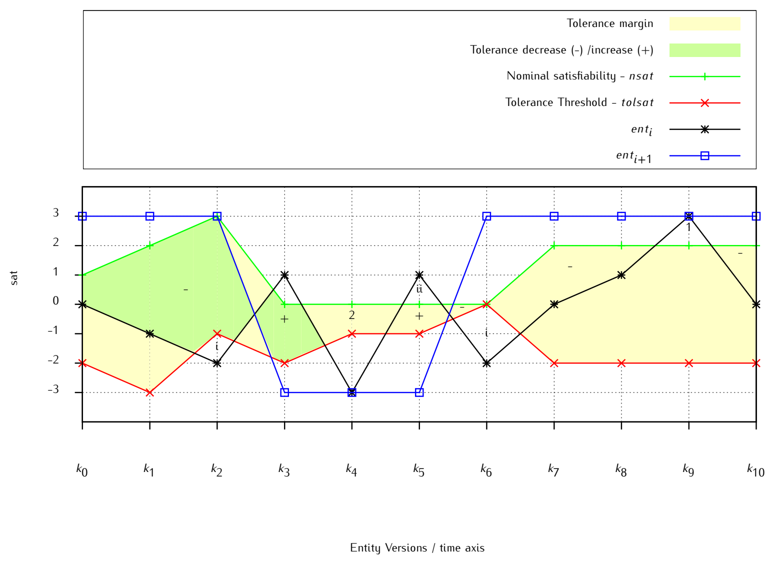


Figure 6. Advanced Satisfiability correlation arbitrary axis /versioning axis.

- mandatory or optional removal of failures.
- focus on specific properties that might be different between evolution steps instead of fusing all properties to evaluate the degradation and the improvement.
- focus on specific evolution intervals at the dynamic evolution axis level in order to reduce the constraints on resiliency.
- provide constraints on the durability of improvement. The resilience from ent_i to ent_j could be required to last over the next generations (no decadence).

4. Model Driven Engineering using the *DREF* framework

4.1. The approach

Our objective is to allow the use of the *DREF* framework at the modelling level no matter what the modelling language is used. The motivation for this objective is clearly to allow modelling notation (DSL- Domain Specific modelling Language) providers to benefit from the *DREF* framework in order to include precisely defined concepts that address dependability and resilience. This will allow users of the modelling notation to explicitly and precisely address dependability concepts. For this objective we propose an approach composed of the following steps:

- Define the meta-model of the targeted DSL using standard meta-modelling tools i.e. ecore equivalent diagrams [13]. The model should include meta concepts derived from the *DREF* meta-model.
- Define properties, entities and observers at the domain level
- Define the evolution axis
- Define the satisfiability functions including the nominal and the tolerance threshold

We begin this section by presenting the *DREF* meta-model. Subsequently, each of these phases is illustrated using a concrete example of a toy DSL in the domain of business process modelling.

4.2. The *DREF* meta-model

In this section, we provide a proposal for introducing *DREF* formal framework elements using meta-modelling techniques. The idea is to propose adequate meta-modelling elements to exploit the definition of a domain specific language that would be targeted by potential future users of *DREF*.
In Figure 7 we provide the model structure of the *DREF* meta-model. The idea is that at the modelling level we request a *DREF* model to be composed of 3+1 categories of models. The first three ones (modes view) are dedicated to the nominal view, the tolerance view, the fail view. The fourth provides the satisfiability view. First, it must be noticed that the modes view could allow one to provide several models per category. This should be defined in the specific meta-model structure. The models provided will propose a specific level of the separation of focus of these views. This means that for a specific DSL the modelling elements for each of these views might be strongly separated while in some other approaches they might be fused. It will be up to the meta-model designer to define the level of separation of concerns that is required.
In the following example, one can observe an approach for defining the level of separation of perspectives concerning a *DREF* extension of some BPMN like meta-model. The example provides a clear separation of modelling elements dedicated to nominal satisfiability, tolerance margin and failures (Figure 18).
In any case, at the semantic level, these views must be implemented for having a means to determine if the satisfiability function should be:

- greater than or equal to the nominal satisfiability (Nominal mode)
- within the tolerance margin (Tolerance mode)
- or lower than the tolerance threshold (Fail mode)

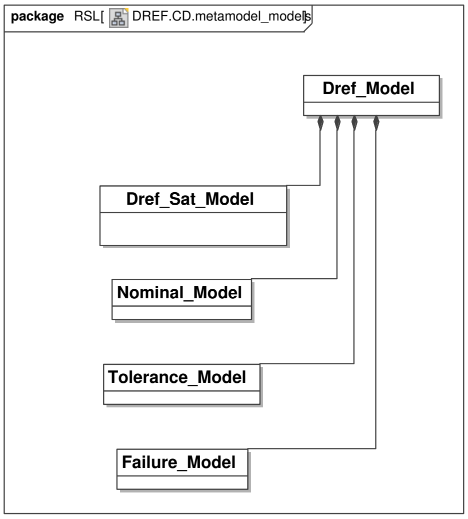


Figure 7. DREF Meta-model - Models view.

Figure 8 represents a proposal for a meta-model fragment. First the meta-model here is incomplete. Second, the approach chosen to define the meta-model elements associated with the concepts defined in the formal definition of the *DREF* framework is very simple so as to be evidently clear. Concerning the incompleteness, many properties are not expressed and will need to be added. These properties should constrain any model satisfying this meta-model and should be compliant with the properties expressed in the formal definition. As an example, the meta-model requires the following properties: unique index values associated with an evolution axis; all tuples of a satisfiability function concerning the same evolution axis; and satisfiability of a property over an entity can only be defined if all observers have this entity and property in their definition domain. For this later property, we would have to parse all the tuples of the *Dref_Satisfiability_tuple* and verify that for all entity and property couples considered there exists a tuple for all the

observers). Furthermore, if we want to impose domain-homogeneous satisfiability functions (cf. see definition 12 and example 20) we should constrain that there is no property for which there is at least one observer capable of evaluating its satisfiability (satisfiability value set to a real number value) and at least one observer incapable of evaluating the satisfiability (satisfiability value set to the undetermined value ⊥).
The satisfiability view is a model of the satisfiability function. In Figure 8 we give an initial proposal for this concept of the *DREF* framework at meta-model level. Any Sat_model defines a satisfiability function. The meta-model elements given in Figure 8 are those that allow for modelling satisfiability functions in extension. This means that a compliant model would contain at least three *Dref_Satisfiability* functions: one for the nominal satisfiability, one for the tolerance threshold and at least one for the effective satisfiability for each evolution axis. Each *Dref_Satisfiability* function is defined by a set of tuples roughly of the form: *<evolution axis name and index, Entity, Observer, Observer Weight, Property, Property eight, a real value or the undetermined value>*.
Of course, depending on the means available for defining such a function, it will be possible to add another model view in which the satisfiability function could be defined (statically or dynamically) differently. As an example, the function could be defined depending on an axiomatisation, the quantitative results of the evaluation of a test set (as it is done in validation testing), or as the quantitative results of the evaluation of a set of theorems (as done in model checking). In any case, the result will always be a satisfiability function that should comply with the meta-model proposed (i.e. the extensional or intentional views of these functions should be coherent).

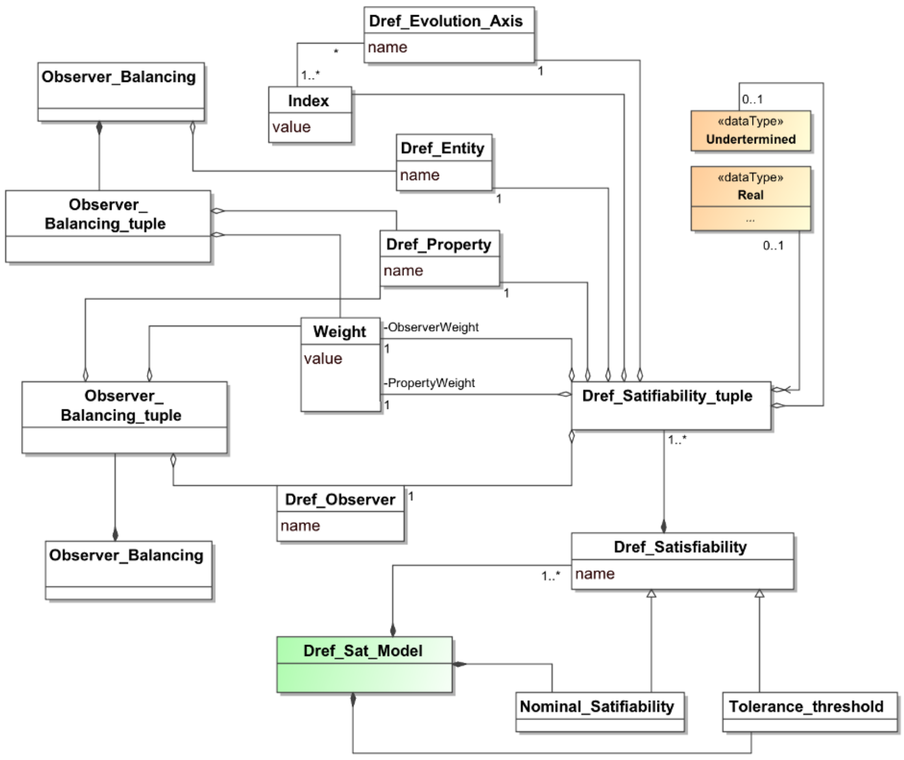


Figure 8. DREF Meta-model - Satisfiability view.

5. Applying the *DREF* framework

In this section, we present a small experimental validation of the approach provided in this article, by engineering a small domain specific language, for modelling *Resilient Business Processes*. We also, discuss other approaches in which the use of the *DREF* framework could be of interest.

5.1. Illustration in the context of Business Process Modelling

In this section we illustrate our approach in the context of a very simple version of a domain specific modelling language dedicated to the modelling of business processes. This DSL is built according to the BPMN (Business Process Modelling Notation) Standard [44].

5.1.1. The BPMN and DREF-BPMN Meta-models

In this illustration, we have chosen to extend a BPMN like DSL with the concepts of the *DREF* framework. In order to proceed, we first have to define the meta-model of the BPMN-like DSL (provided in Figure 9). It indicates that a specification can contain three types of models: a BPMN model (example of an instance given in Figure 11), a constraint model (instance given in Figure 14) and two models subtypes of class diagram: concept model (instance given in Figure 12) and interface model (instance given in Figure 13). Furthermore, a BPMN model is a simplified standard BPMN [44] (i.e. pools, lanes, activities or tasks). The simple particularity is that we attach pre/post expressions using the Object Constraint Language (OCL) to tasks, thus providing an axiomatic specification of tasks (see Figure 14).

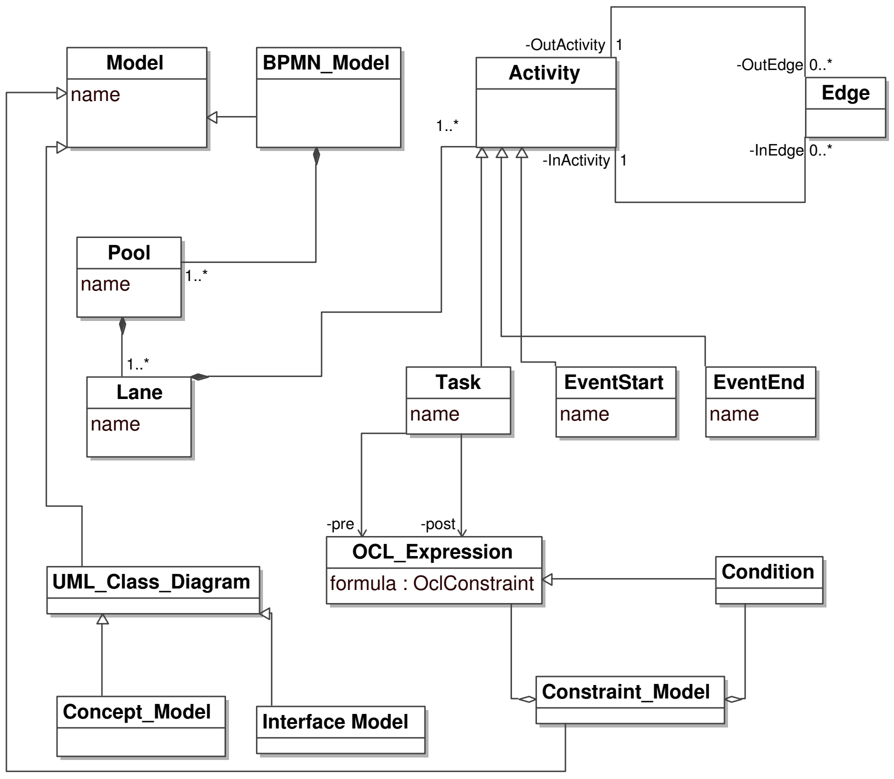


Figure 9. Simplified BPMN Meta-model.

We then have to define an extended BPMN meta-model demonstrating its integration with the *DREF* meta-model. In Figure 10, we illustrate such an extended meta-model. In the context of this simple example, we have defined a *DREF* model for which the nominal, tolerance and failure models are defined as BPMN models. Of course, the constraints provided for each of theses three models will define what is allowed to be included in these models[14]. It

[14] *These constraints are not provided in this article since the objective is not to focus on the DSL definition but on the use of DREF for extending DSL.*

is important to note that the extension proposed concerns only the structural part which is related to the models view of the *DREF* meta-model (see Figure 7). Concerning the satisfiability functions, the example given in this section does not provide the associated meta-models. The idea is nevertheless the same i.e. for each property and entity (in this example, business process instances) a satisfiability value is provided (see the graphical representation in Figure 16).

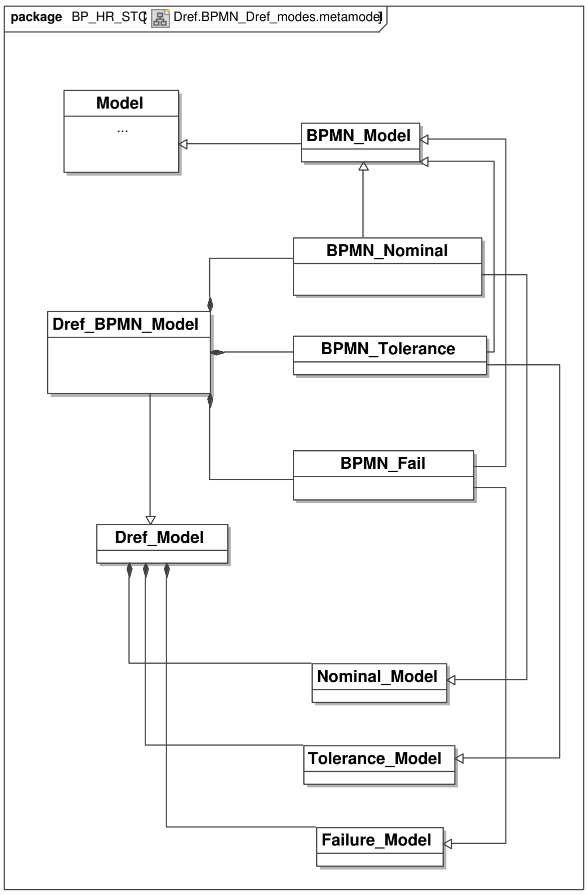


Figure 10. DREF-BPMN Meta-model.

5.1.2. The Nominal Mode

We consider a simple Human Resources (HR) Department. The HR department has one business process which manages short term contracts (STC) for specific missions. Let us have the following informal requirements description: HR department has as a goal to process the STC recruitments for the different services of the company. Each time a service has a position to fill, he provides the position description to the HR such that HR can start the process. HR publishes the position using external diffusion (such as journals, web, national recruitment services). Applications are received until the position is closed i.e. when at least 2 applicants applied. All applicants are interviewed and then the HR selects the best candidate. The contract is signed with the selected candidate. Finally, company service and unselected applicants are notified.

According to our meta-model, this BPMN model for the nominal mode is made of a process model (cf. Figure 11), a concept model (cf. Figure 12 and 13) and a constraint model (cf. Figure 14). In Figure 11, we provide the BPMN model of this STC contract BP of the HR department[15]. This BP model should be considered as the process that the HR must apply in case of STC management.

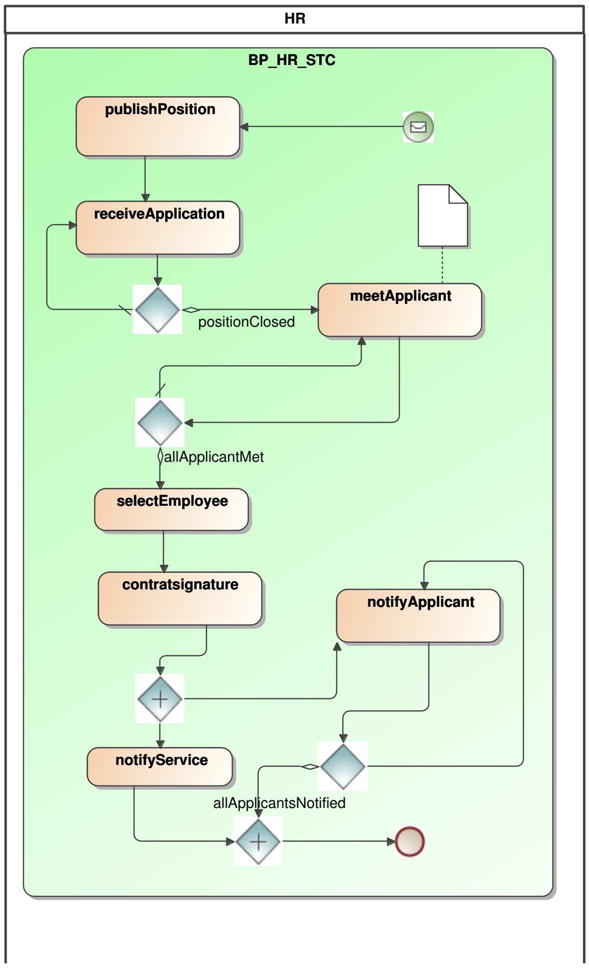


Figure 11. HR Short Term Contract BP - Nominal mode - process model.

According to our meta-model (Figure 10), the activities are related to data which are modelled in the concept model (Figure 12). This data model more precisely describes the concepts, which are mentioned in the informal requirements (of course this should be performed in an iterative process involving all the concerned stakeholders).

In order to specify more precisely the activities of the BP, we provide for each of its activities (listed in the interface view of Figure 13) a pre/post axiomatic specification using OCL [43]. An example of such a specification is provided in Figure 14. In this specification, the pre-condition of the *meetApplicant* activity indicates that the HR department has not previously interviewed the candidate (if this is not the case, the outcome of this activity is not known and the fault will not be attributed to the HR). The post-condition, indicates that the HR registers the fact that this applicant has been interviewed.

[15] *Of course this BPMN model is proposed solely to illustrate the proposed approach. It must be considered only as such by the reader.*

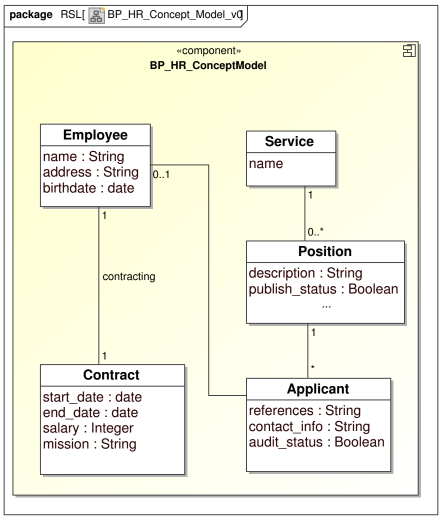


Figure 12. HR Short Term Contract BP - Nominal mode - Concept model.

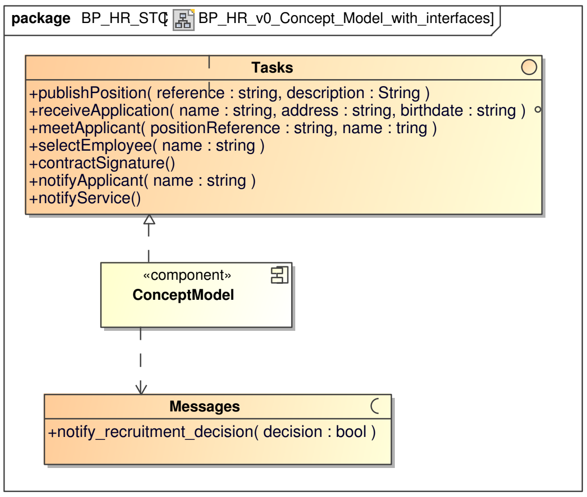
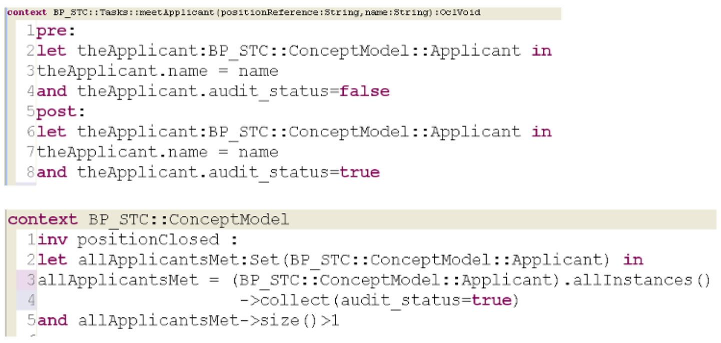

package BP_HR_STC[BP_HR_v0_Concept_Model_with_interfaces]
Tasks
+publishPosition(reference : string, description : String)
+receiveApplication(name : string, address : string, birthdate : string)
+meetApplicant(positionReference : string, name : tring)
+selectEmployee(name : string)
+contractSignature()
+notifyApplicant(name : string)
+notifyService()
«component»
ConceptModel
Messages
+notify_recruitment_decision(decision : bool)

Figure 13. HR Short Term Contract BP - Nominal mode - Process Activities and Messages.

Pre- and post-conditions are part of the constraints' model as well as the conditions used in the BP model for gateways. In the same model, the *positionClosed* logical property is specified as a Boolean operation that is true *iff* the total number of applicants met is more than one.

5.1.3. Entities, Observers, Properties and Balancing

Having defined the nominal view of the business process, we now define Entities, Observers, Properties and Balancing in the following way:



```
context BP_STC::Tasks::meetApplicant(positionReference:String,name:String):OclVoid
pre:
let theApplicant:BP_STC::ConceptModel::Applicant in
theApplicant.name = name
and theApplicant.audit_status=false
post:
let theApplicant:BP_STC::ConceptModel::Applicant in
theApplicant.name = name
and theApplicant.audit_status=true
```

```
context BP_STC::ConceptModel
inv positionClosed :
let allApplicantsMet:Set(BP_STC::ConceptModel::Applicant) in
allApplicantsMet = (BP_STC::ConceptModel::Applicant).allInstances()
                        ->collect(audit_status=true)
and allApplicantsMet->size()>1
```

Figure 14. HR Short Term Contract BP - Nominal mode - constraints model extract.

- *Entities*: We propose to consider real business processes executions as entities[16]. A real business process is given in terms of a description of what has been executed by the HR department. In order to continue in a consistent way with our model driven engineering approach, a real business process will be given as a model. In the case of sequential semantics where the participants in the BP are not distinguishable, they should correspond to a set of directed graphs representing the sequence of activities and messages labelling links between nodes representing the BP state[17]. To simplify, we will just model them as a an ordered set of events being activity or message identifiers. As an example, we consider the four BP instances modelled as oriented graphs[18] in Figure 15.

- *Observers*: Each of these BP instances are observed by only one observer who is the Quality Control Officer (QCO) of the company.

- *Properties*: The QCO is interested in observing the entities using the model that has been given to the HR department. In this very simple case, we consider that the property is provided by the BP Process Models' view (the conjunction of the models provided in Figures 11, 12, 13 and 14).

- *Balancing*: In this very simple illustration we have one observer and one property. We thus have a default balancing such that the weight of the observer and the property are equal to 1.

5.1.4. Evolution and Observation axis

We define two axes for the purpose of illustrating the *DREF* framework in model driven engineering and development processes:

- An evolution axis (*ev*) has two indexes (1 and 2) representing one evolution step of the HR service. While in the first one the HR employee activities would be evaluated according to a non fault-tolerant business process (modelled in Figure 17). The second version of the HR department introduces a tolerance margin (described in Figure 17).

- An observation axis (*obs*) with four indexes (1 to 4) to denotes the 4 observation points is shown in Figure 15. They represent four completed recruitment processes executed by the actual HR Service.

[16] *remind that the choice of what are the entities of interest depends on the objectives for the dependability and resilience evaluation*

[17] *This is the usual model used to provide the description of the actual execution of behavioural models with or without states. See [11] in the case of Petri nets for behaviour and algebraic specifications for data.*

[18] *Graphs in which nodes are business process states. Edges from a node n_1 to a node n_2 labelled with $t(a_1, \ldots, a_n)$ indicate that the task $t(a_1, \ldots, a_n)$ can be executed from state n_1 leading to state n_2.*

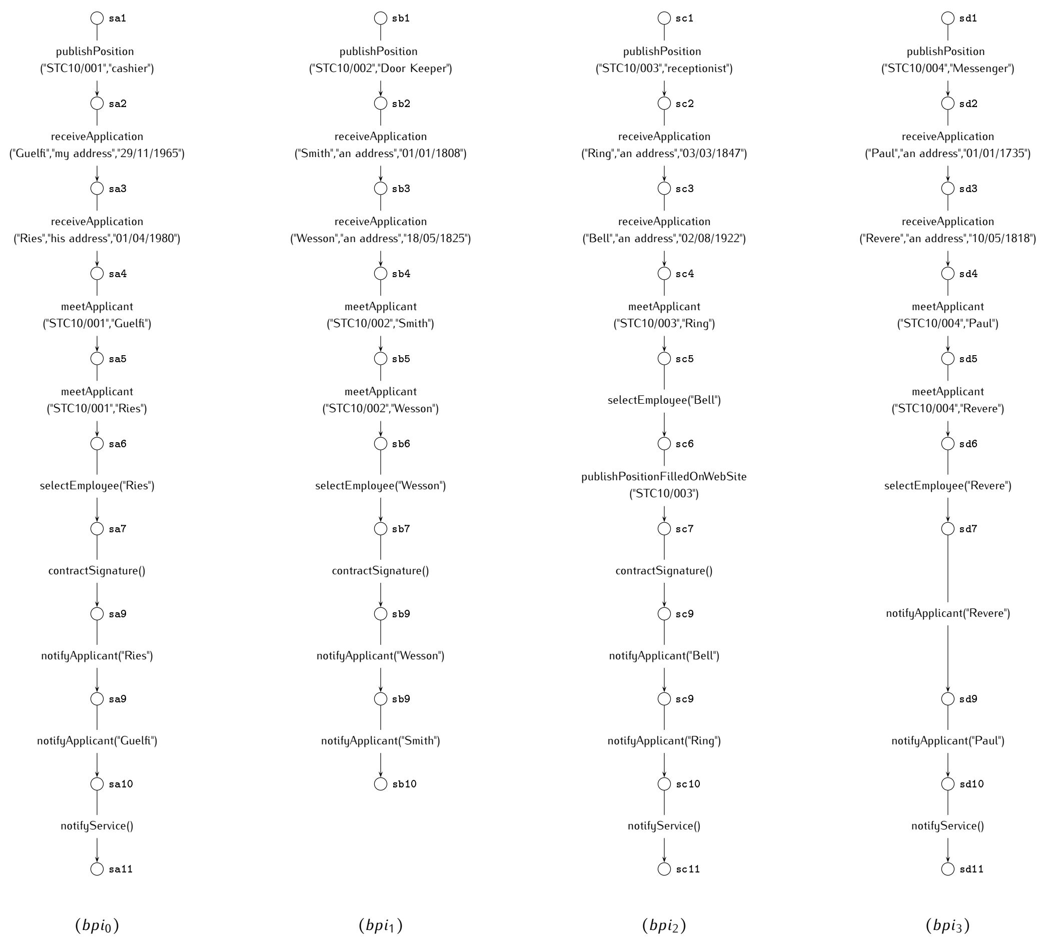

sa1
publishPosition ("STC10/001","cashier")
sa2
receiveApplication ("Guelfi","my address","29/11/1965")
sa3
receiveApplication ("Ries","his address","01/04/1980")
sa4
meetApplicant ("STC10/001","Guelfi")
sa5
meetApplicant ("STC10/001","Ries")
sa6
selectEmployee("Ries")
sa7
contractSignature()
sa9
notifyApplicant("Ries")
sa9
notifyApplicant("Guelfi")
sa10
notifyService()
sa11
(bpi0)
sb1
publishPosition ("STC10/002","Door Keeper")
sb2
receiveApplication ("Smith","an address","01/01/1808")
sb3
receiveApplication ("Wesson","an address","18/05/1825")
sb4
meetApplicant ("STC10/002","Smith")
sb5
meetApplicant ("STC10/002","Wesson")
sb6
selectEmployee("Wesson")
sb7
contractSignature()
sb9
notifyApplicant("Wesson")
sb9
notifyApplicant("Smith")
sb10
(bpi1)
sc1
publishPosition ("STC10/003","receptionist")
sc2
receiveApplication ("Ring","an address","03/03/1847")
sc3
receiveApplication ("Bell","an address","02/08/1922")
sc4
meetApplicant ("STC10/003","Ring")
sc5
selectEmployee("Bell")
sc6
publishPositionFilledOnWebSite ("STC10/003")
sc7
contractSignature()
sc9
notifyApplicant("Bell")
sc9
notifyApplicant("Ring")
sc10
notifyService()
sc11
(bpi2)
sd1
publishPosition ("STC10/004","Messenger")
sd2
receiveApplication ("Paul","an address","01/01/1735")
sd3
receiveApplication ("Revere","an address","10/05/1818")
sd4
meetApplicant ("STC10/004","Paul")
sd5
meetApplicant ("STC10/004","Revere")
sd6
selectEmployee("Revere")
sd7
notifyApplicant("Revere")
sd9
notifyApplicant("Paul")
sd10
notifyService()
sd11
(bpi3)

Figure 15. Actual BP instances observed at HR Department.

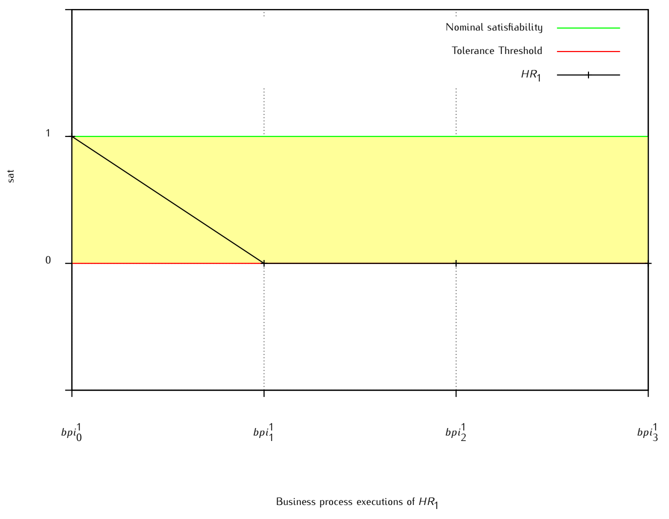
5.1.5. The Satisfiability view

Once the entities distributed along the given evolution axes, properties and observers (with or without balancing) are defined, we have to provide the satisfiability view as a model instance of the meta-model provided in section 4.2. In the case of our toy example, we have one observer (HRD as the director of the HR service), one property as the business process model given in Figure 11 and eight entities $(bpi_0^1, ..., bpi_3^2)$[19] .
Satisfiability view is then given as a set of tuples defining the satisfiability functions. In the context of our case study, we represent them by the two plots given in Figure 16.
In our illustrations, the values of the satisfiability functions are determined using the nominal and tolerance modes provided in the corresponding models given in Figure 17. The satisfiability function is such that:

- $sat(bpi_i^j) = 1$ *iff* bpi_i^j belongs to the business process instances that are "instances" of the Nominal model (i.e. complies with the Nominal mode definition)[20].

[19] *To simplify we have considered that the same business processes were executed by the two evolutions of the HR service. Thus* bpi_i^1 *and* bpi_i^2 *are the same for* $i = 0, ..., 3$.
[20] *Of course, if we want to be more formal, we would have to define the mathematical semantics of the nominal view as a semantic function from the Nominal models to the powerset of instance models. This would then partition the set*

- $sat(bpi_i^j) = 0.5$ *iff* bpi_i^j belongs to the business process instances that are "instances" of the Tolerance model (i.e. it complies with the Tolerance mode definition).
- $sat(bpi_i^j) = 0$ *iff* bpi_i^j is any other business process instance.



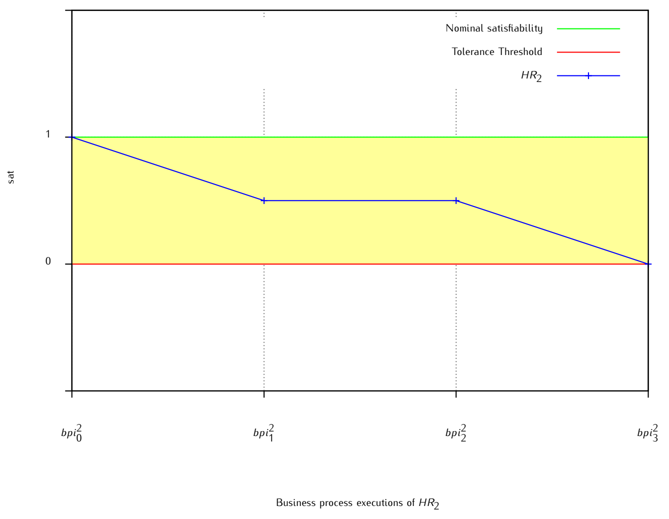


Figure 16. Satisfiability for HR version 1 & 2.

In Figure 18, we include the Failure view(s), which is used here only to model a subset of the business process instances for which the satisfiability function returns a failure (below the tolerance margin). Depending on the *DREF* extension targeted, either this view could be considered as fully characterising the failures or only a subset. In the first case, any business process instance not in the nominal, tolerance, or failure sets should either not be in the definition domain of sat or the sat value should be $\perp$. In the second case, the view should be used to document a subset of failing business

of instance models into two categories: 1) the instance models satisfying the model (satisfying entities) and 2) those failing the model the other (failing entities).

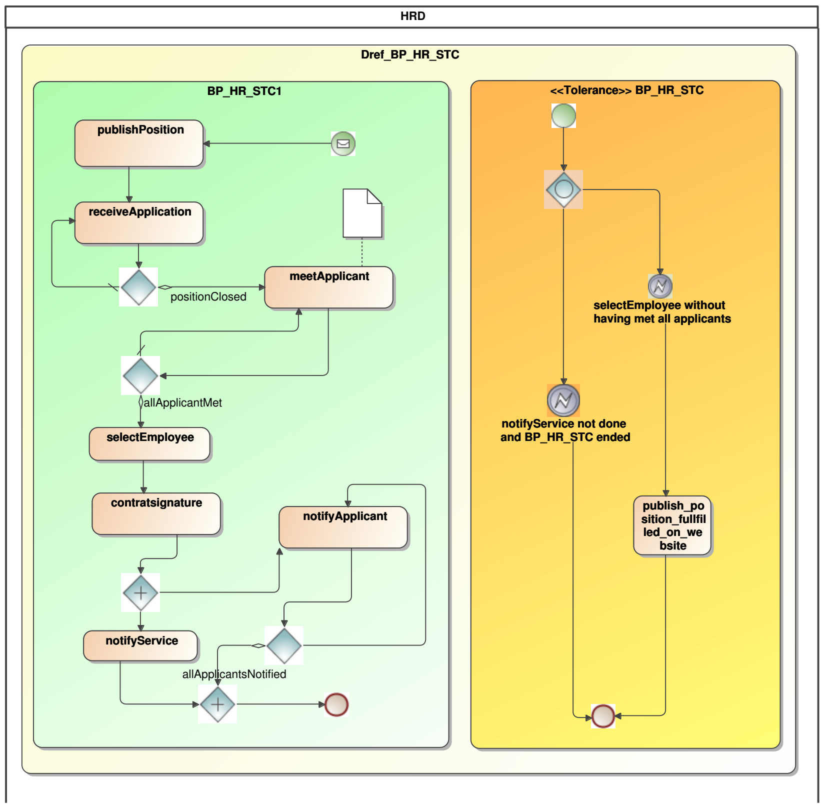


Figure 17. HR Short Term Contract BP - process model with tolerance modelling.

process instances. In the case of failure modes, the view could be used to partition the failure margin in failure modes (the red variants). This can be generalised to the tolerance view, which would provide a means to model the degradation modes[21].

5.1.6. Tolerance, failure and Resilience

Given the satisfiability functions, we can now analyse the basic dependability elements, which are failure and tolerance, and the $T-Resilience, F-Resilience, TF-Resilience$ over the evolution axes. In our case study, we have $\Delta qFail_{1,2} = 3-1 = 2$ and $\Delta tol_{1,2} = 13-12 = 1$. Thus we can say that there is T-Resilience and F-Resilience (and so TF-resilience) in the evolution process from HR_1 to HR_2.

5.2. Other Approaches for *DREF* Satisfiability functions

In the context of safety critical system development, we have addressed the problem of defining a development process that would improve the dependability of engineered software along the versioning evolution axis. The context for this problem was the development of embedded safety critical software for a car's airbag opening system. The approach was based on a validation using testing [24]. Thus each software version was validated based on the results of a set of test cases. A constraint was to ensure non-regression in the versioning. The meta-models for the nominal view were provided in terms of protocol state machine and class diagrams with OCL constraints. After a thorough analysis of the application

[21] *For coloured screens or printers it is the orange variants and the middle case for the others*

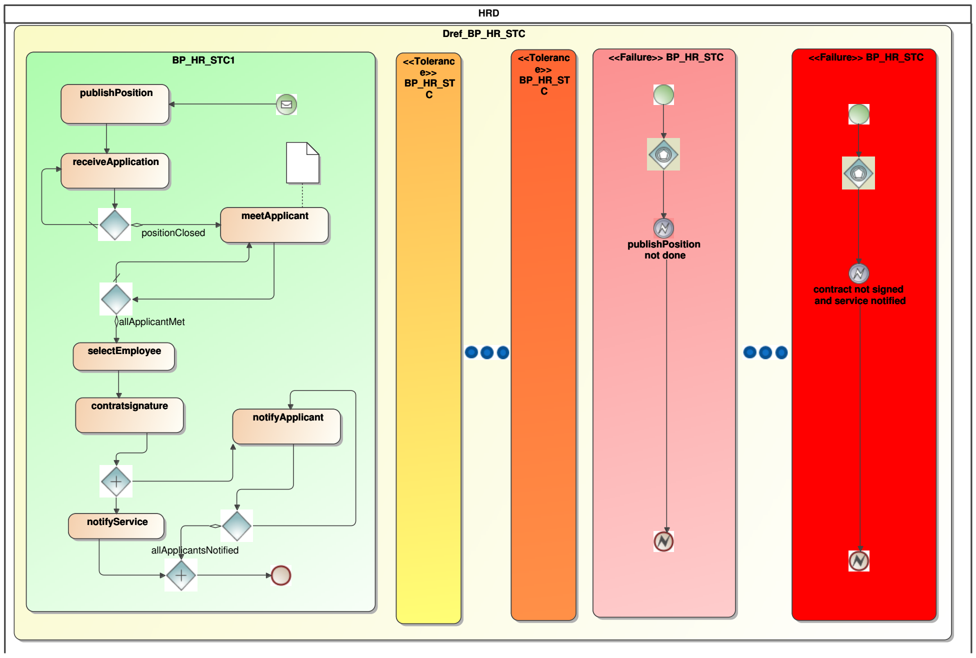


Figure 18. HR Short Term Contract BP - process model with tolerance and Failure modelling.

domain, it has been concluded that the validation would be dependent on 5+2 views (cf. Figure 19). Each of these views would then select a set of test cases to be used to define the satisfiability value of the software version. Even if it would be appropriate, the first approach used did not define a balancing, neither between the views nor between the test cases.

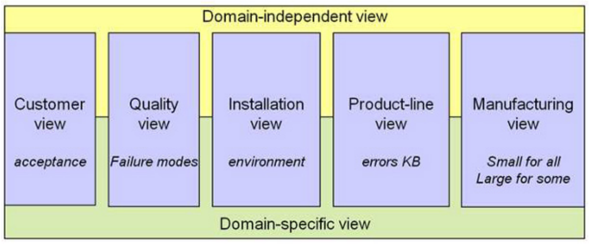


Figure 19. Views.

Using test results as a mean to compute the satisfiability is a very valuable and pragmatic approach for *DREF*. From a similar perspective, model checking techniques could also be considered. Practical use of the *DREF* framework is currently limited to the field of DSL development (as illustrated by the case study). We are currently conducting two additional experiments to better assess the framework. One experiment in the field of operational resilience [37] uses entities as algebraic Petri net models [47], properties as invariants regarding places in the APN, and satisfiability as logical functions computed using the model checker AlPiNA [12]. A second experiment was performed in the context of architecture description languages [48]. The aim was to improve the AADL modeling language with the *DREF* concepts, thus engineering an architecture description language for resilient architectures. Those experiments provide a practical assessment of the usefulness of the approach presented in this article for the specific aims targeted.

6. Related Work

Resilience Definition With respect to the definition of resilience that derives from the dependability and the adaptive systems communities in information and communication science, many representative researchers [8, 14, 33, 52] share the same definition[22]. From their perspective, resilience is initially defined as "the persistence of service delivery that can justifiably be trusted, when facing changes" and mainly regarded as equivalent to fault-tolerance. However, its use is considered emphasize the notion of "unforeseen events" and to include the effects of evolution through the "change" concept. While no formal or informal definition of resilience is proposed here, the authors' intuition regarding resilience shows that the *DREF* formal conceptual framework could be used to provide a coherent and more precise definition of resilience.
Unforeseen events could be considered as events chosen to define an observation axis over which degradation and failures are quantified. In this case, the idea could be that an observation axis for the entity can be partitioned into foreseen events (i.e. for which the entity can cope with according to a explicitly pre-defined way) and unforeseen events (i.e. events for which the entity can cope with but in an way not determined explicitly). The foreseen or unforeseen nature of these events has no semantic relationships with the values for the satisfiability function for the selected properties. This means that there might be satisfiability, degradation or failure. The two dimensions emphasised are the capability to provide resilience and the need for the existence of a justification of this provision. In *DREF*, the first aspect is of course ensured by the evolution of the satisfiability function while the second is provided by the explicit definition of the satisfiability function over the evolution axis. This is considered as the capability of our framework to provide a justification that resilience is ensured.
Nevertheless, we should remark that only focusing on the persistence of service delivery is too restrictive to characterise the concept of resilience. In *DREF* this would mean that the satisfiability function should be always over the nominal satisfiability, never decreasing and furthermore, using a binary quantification (only correct service delivery or incorrect). In our approach, there is a fundamental difference that forbids us to consider resilience as a synonym of fault-tolerance. It is the explicit consideration of an evolution axis for satisfiability. Thus if we want to related resilience to fault-tolerance, resilience could be linked to the evolution of fault-tolerance capabilities over an evolution axis.
If we widen our study to other fields (scientific or not) [25, 28, 41, 55], we will notice that the set of keywords used to define resilience is quite stable. Those keywords are in a way or another related to adaptation, recovery, improvement, fault-tolerance, reliability, resistance, robustness. Whatever the differences, we can say that for each notion of resilience proposed in this literature, the *DREF* framework can be used to provide more rigour by allowing to formally define each notion of resilience using the framework.
Model Driven Engineering and resilience We can also find many studies in which the integration of dependability and evolution concepts are introduced at the modelling or meta-modelling level. This is done at different levels of formalisation, going from informal i.e. natural languages to formal i.e. mathematical languages. In [39], the authors propose to explicitly add, at the requirement level some non-functional properties directly related (at the abstract level) to dependability. Metrics for quantification of these attributes are also informally introduced in accordance with our approach. Nevertheless, the attributes used to characterise resilience are completely different to our approach since resilience is defined as availability, reliability and "assurance".
In [14] the authors explicitly foresee the need for such a model based approach: "to deal with the challenges of adaptation we envisage a model-driven development, where models play a key role throughout the development ... In fact, models can support estimation of system' status, so that the impact of a context change can be predicted. Provided that such predictions are reliable, it should be possible to perform model-based adaptation analysis as a verification activity.". Our framework and its integration into a model driven engineering perspective is partly designed for this purpose. In addition, we widen the problem space to any software engineering methodology for resilient systems using modelling techniques in an approach similar to that followed in [20].
If we focus on business process modelling and evolution, we cite [36] that presents an approach for managing several variants of business processes to which quantification is attached and used to evaluate the evolution between variants.

[22] *With regards to the other definitions of resilience in information and communication science, we have noticed, that for the most part they are often poorly specified using a cloud of keywords that are sometimes not consistent with the concepts of dependability. Therefore, we did not consider useful to present them even if numerous*

This approach is fully compatible with the *DREF* framework even if the business process variants are not directly related to resilience.
Validation, conformance (or compliance) evaluation or certification: Our framework is fundamentally based on the notion of satisfiability. If we focus on this notion and its use in information and communication science, we rapidly see that it is related to quality and its validation, conformance (or compliance) evaluation, assessment or certification. These are activities for which an extensive set of results in research and industry have been developed in many engineering fields including, of course, information and communication science [1, 2, 10, 27, 29]. Concerning software product quality assessment, the assessment corresponds to different levels of precision as an evaluation of *conformance properties* on *artifacts* using a *conformance analysis* technique. This technique, when formal, is often defined using metrics or measures[23] [30, 50, 51]. An extensive bibliography exists, which, in the general case, is compatible with the *DREF* framework. Either at the conceptual level for quality assessment from a customer point of view [15] or from a more formal approach for the static or dynamic evaluation of product quality for control and improvement [31]. We consider that the *DREF* framework is at a higher level of abstraction and it has been designed to be coherent with the concepts and approaches used in software quality measurement and management [18, 19, 49, 54, 56]. Do be refined in the directions proposed by this literature, it will need to couple the satisfiability function definition with some adequate metrics. These metrics can either be defined using software analysis techniques that can be static or dynamic (e.g. monitoring, test results, model checking).

7. Critical Analysis and Perspectives

The aim of this work is to solve a software engineering problem by a proposing a formal conceptual framework allowing for domain specific language engineering for model driven development of dependable systems. As a consequence, we can draw the following remarks that help the reader understand our focus and the limitations of our work:

- The framework is deliberately generic and is designed as such. Its use in specific domains will necessitate its refinement.

- The aim of the framework is not to contain, in its abstract version, the means to define all the metrics for each of the dependability attributes. The framework, should allow for being refined to do so. With such a perspective, probabilist computations of satisfiability should be studied in the near future. Metrics that give probabilistic satisfaction for properties of interest will imply specific definitions of the global satisfiability as well as the resilience property. Nonetheless, there will be failure or degradation situations that we want to be resilient to and for which the probabilistic approach is not adequate.

- The framework is not a solution to make a non resilient system resilient. It simply allows, if metrics are available, to integrate them in a model driven approach to the development of resilient systems.

- A major aspect of the approach resides in defining "useful" satisfiability functions. This is of course a field of improvement of the *DREF* framework and the first experiment using boolean quantification has already been proposed in [37]. As stated in the previous section, the next step will be to define metrics based on test case specifications and quantifications. These results will be applied to concrete cases based on our expertises in trials at the court of justice in which we have conducted several conformance evaluations. In these expertises, the conformance question is always asked by the judge to the expert. Thus defining observers, properties and weights is a crucial preliminary action to be conducted. Then determining the satisfiability of those properties based on the artifacts provided like documentation (such as specifications, design, test plan and test results, source code) in a reasonable time frame is a sensible task. We also encountered several times the need to compare program versions w.r.t. expected improvements. All this makes the *DREF* framework a good candidate to be used as a tool to support those expertises.

[23] *Our approach adheres to the importance of measurements and metrics as as been well established by Lord Kelvin in [32]. In addition, one can consult [22] for a discussion on the terminology and its use.*

8. Conclusion

In this article, we have introduced an initial version of a formal framework (*DREF*) that precisely defines the fundamentals concepts used to define dependability and resilience of ICT systems. This framework has been defined using set theory at a chosen abstraction level in order to cover the current advances in the terminology of dependable and resilient systems engineering. The proposed framework has been designed to be useful for ICT system model driven engineering. To this extent, a meta-model has been proposed for the framework and a validation of the approach using the *DREF* framework for modelling language engineering has been provided. Many possibilities are now being considered to continue this research. One direction might be to integrate the framework with meta-data approaches [21] to quality of service for expressing properties, satisfiability. Another possibility might be a complete proposal for the production of ICT systems models' generation over an evolution axis in terms of model transformations using an imperative or axiomatic model transformation language. It could be developed to be integrated into a complete development methodology based on the refinement of formal specifications together with a formal verification framework based on model checking for satisfiability evaluation [23, 35]. A more theoretical approach could be to re-engineer the definition of the framework using mathematical statistics which seems to be a promising direction especially for ICT systems compliance assessment particularly in the field of certification.

Acknowledgments

The author would like to thank all the colleagues met mainly in the context of the ERCIM working group SERENE - Software Engineering for Resilient Systems (http://www.ercim.eu/activity/workgroup) which have motivated and inspired the content of this article. Many thanks to Tonie M. van Dam, Benoit Ries, Gilles Perrouin, Alfredo Capozucca, Ayda Saidane and Levi Lucio for their valuable comments on the successive versions of this entity that hopefully resulted in a resilient writing process compatible with satisfiability and within the tolerance margin of the journal reviewers.

References

[1] IEEE standard for a software quality metrics methodology. Tech. rep. 1998

[2] Al-Qutaish R.E., Quality models in software engineering literature, An analytical and comparative study, Journal of American Science, Marsland Press, Michigan, USA, 2010, Vol. 6, No. 3, 166-175

[3] Atkinson C., Kuhne T., Model-driven development: A metamodeling foundation. IEEE Software, 2003, 20(5), 36-41, http://csdl.computer.org/comp/mags/so/2003/05/s5036abs.htm

[4] Avgeriou P., Guelfi N., Resolving architectural mismatches of cots through architectural reconciliation. In: Proceedings of the International Conference on COTS-Based Software Systems (ICCBSS), Springer-Verlag, 2005

[5] Avgeriou P., Guelfi N., Perrouin G., Evolution through architectural reconciliation. In: Proceedings of the workshop on "Software Evolution Through Transformations" (SETra) 2004. Electronic Notes in Theoretical Computer Science, 2004, Vol. 127(3), 165-181

[6] Avizienis A., Laprie J.C., Randell B., Fundamental concepts of dependability. Tech. rep., Computer Science Department, University of California, Los Angeles, USA, 2001

[7] Avizienis A., Laprie J.C., Randell B., Landwehr C.E., Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Sec. Comput., 2004, 1(1), 11-33

[8] Avizienis A., Laprie J.C., Randell B., Landwehr C.E., Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable Sec. Comput., 2004, 1(1), 11-33

[9] Black P.E., Windley P.J., Verifying resilient software, 1997, 262-266

[10] Boehm B.W., Brown J.R., Lipow M., Quantitative evaluation of software quality. In: ICSE, 1976, 592-605

[11] Buchs D., Guelfi N., A formal specification framework for object-oriented distributed systems. IEEE Trans. Software Eng., 2000, 26(7), 635-652

[12] Buchs D., Hostettler S., Marechal A., Risoldi M., Alpina, A ymbolic model checker. In: Petri Nets, Lecture Notes in Computer Science, Vol. 6128, Springer, 2010

[13] Budinsky F., Brodsky S.A., Merks E., Eclipse Modeling Framework. Pearson Education, 2003
[14] Cheng B., al., Software engineering for self-adaptive systems, A research roadmap. In: B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee (eds.) Software Engineering for Self-Adaptive Systems, Lecture Notes in Computer Science, Vol. 5525, 1-26, Springer, 2009
[15] Chulani S., Ray B., Santhanam P., Leszkowicz R., Metrics for managing customer view of software quality. 189-198, 2003
[16] Dearnley P.A., An investigation into database resilience. Comput. J., 1976, 19(2), 117-121
[17] Erl T., Service-Oriented Architecture, Concepts, Technology, and Design. Prentice Hall, 2006
[18] Fenton N., Software measurement, a necessary scientific basis. Software Engineering, IEEE Transactions on, 1994, 20(3), 199-206
[19] Fenton N.E., Pfleeger S.L., Software Metrics, A Rigorous and Practical Approach. International Thompson Computer Press, 1996
[20] Fleurey F., Dehlen V., Bencomo N., Morin B., Jézéquel J.M., Modeling and validating dynamic adaptation. In: MoDELS Workshops, 97-108, 2008
[21] DiMarzo Serugendo G., Fitzgerald J., Romanovsky A. and Guelfi N.: Towards a Metadata-Based Architectural Model for Dynamically Resilient Systems. In: SAC'07, March 11-15, Seoul, Korea, ACM Press, 2007
[22] GarcÃa F., Bertoa M.F., Calero C., Vallecillo A., RuÃz F., Piattini M., Genero M., Towards a consistent terminology for software measurement. Information and Software Technology, 2006, 48(8), 631-644
[23] Guelfi N., Pelliccione P., Muccini H., Romanovsky A., Software Engineering of Fault Tolerant Systems. World Scientific Publishing Co., 2007
[24] Guelfi N., Ries B., Selection, evaluation and generation of test cases in an industrial setting, a process and a tool. In: Testing, Academic & Industrial Conference, 47-51, IEEE, Windsor, UK, 2008
[25] Hamel G., Valikangas L., The quest for resilience. Harvard business review, 2003, 81(9), 52-63
[26] Harel D., Statecharts, A visual formalism for complex systems. Science of Computer Programming, 1987, 8(3), 231-274
[27] Heck P., Klabbers M., Eekelen M., A software product certification model. Software Quality Control, 2010, 18(1), 37-55
[28] Hollnagel E., Woods D., Leveson N. (Eds.), Resilience Engineering, Concepts And Precepts. Ashgate Publishing, 2006
[29] (ISO) International Standard ISO/IEC 9126. information technology: Software product evaluation, Quality characteristics and guidelines for their use, 1991
[30] Kan S.H., Metrics and Models in Software Quality Engineering. Addison-Wesley Longman Publishing Co., Inc., 2002
[31] Kanoun K., A measurement-based framework for software reliability improvement. Annals of Software Engineering, 2001, 11, 89-106
[32] Kelvin W.T., Popular lectures and addresses [microform] / by Sir William Thomson, Macmillan, London, New York, 1889
[33] Laprie J.C., From dependability to resilience. In: Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks, DSN - Fast Abstracts. IEEE/IFIP, 2008
[34] Ledyayev R., Architectural framework for product line development of dependable crisis management systems. master thesis, Universite Henri Poincare Nancy 1, 2009
[35] Liu Y., Muller S., Xu K., A static compliance-checking framework for business process models. IBM Systems Journal, 2007, 46(2), 335-362
[36] Lu R., Sadiq S., Governatori G., On managing business processes variants. Data Knowl. Eng., 2009, 68(7), 642-664
[37] Lúcio L., Guelfi N., A precise definition of operational resilience. Tech. Rep. TR-LASSY-11-02, Laboratory for Advanced Software Systems, University of Luxembourg, 2011
[38] Ludewig J., Models in software engineering. Software and System Modeling, 2003, 2(1), 5-14
[39] Mostert D.N.J., von Solms S.H., A technique to include computer security, safety, and resilience requirements as part of the requirements specification. J. Syst. Softw., 1995, 31(1), 45-53
[40] Naur P., Randell B., Software engineering report of a conference sponsored by the NATO science committee Garmisch Germany 7th-11th October 1968, 1969, http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

[41] Nemeth C.P., Resilience Engineering, The Birth of a Notion, vol. 1, chap. 2, 3-9, Ashgate Publishing, 2008

[42] OMG, Uml 2.0 infrastructure specification. Tech. rep., Object Management Group, 2003

[43] OMG, Ocl 2.0 specification. Tech. rep., Object Management Group, 2005

[44] OMG, Business Process Modeling Notation (BPMN), Version 1.2, 2009, http://www.omg.org/spec/BPMN/1.2/PDF

[45] Pohl K., Böckle G., van der Linden F.J., Software Product Line Engineering: Foundations, Principles and Techniques, 1 edn. Springer, 2005

[46] Randell B., Software engineering, As it was in 1968. In: ICSE, 1-10, 1979

[47] Reisig W., Petri nets and algebraic specifications. Theor. Comput. Sci., 1-34, 1991

[48] Saidane A., Dref resiliency and security aspects in the sae architecture analysis and design language (aadl). Tech. Rep. TR-LASSY-10-07, University of Luxembourg, 2010

[49] Schneidewind N., Measuring and evaluating maintenance process using reliability, risk, and test metrics. Software Engineering, IEEE Transactions on, 1999, 25(6), 769-781

[50] Schneidewind N.F., Software metrics model for quality control. In: 4th IEEE International Software Metrics Symposium (METRICS 1997), November 5-7, 1997, Albuquerque, NM, USA, 127-136, IEEE Computer Society, 1997

[51] Schneidewind N.F., Body of knowledge for software quality measurement. Computer, 2002, 35, 77-83

[52] Simoncini L., Resilient computing, An engineering discipline. Parallel and Distributed Processing Symposium, International, 1, 2009

[53] Svobodova L., Resilient distributed computing. IEEE Trans. Software Eng., 1984, 10(3), 257-268

[54] Tian J., Quality-evaluation models and measurements. Software, IEEE, 2004, 21(3), 84-91

[55] Ungar M., Resilience across cultures. Br. J. Soc. Work., 2008, 38(2), 218-235

[56] Weerahandi S., Hausman R., Software quality measurement based on fault-detection data. Software Engineering, IEEE Transactions on, 1994, 20(9), 665-676

[57] Wirsing M., Algebraic specification. In: Handbook of Theoretical Computer Science, Volume B, Formal Models and Semantics B, 675-788, 1990