CONSTRATNENTS FOR AVOIDING SYSML MODEL INCONSISTENCIES

Cristian Capozucco
cco18001@student.mdh.se

Examiner: Antonio Cicchetti
Mälardalen University, Västerås, Sweden

Supervisors: Federico Ciccozzi
Mälardalen University, Västerås, Sweden
Jan Carlson
Mälardalen University, Västerås, Sweden
Davide Di Ruscio
Università degli Studi Dell’Aquila, L’Aquila, Italy

June 18, 2019
Abstract

Models are used in multiple phases of a development process for several purposes. However, models may present inconsistencies. This is often due to the modelling language leaving a certain degree of freedom to the modeller or the tool, implementing the language and used by the modeller, not providing enough support for identifying possible inconsistencies. This thesis identifies a certain number of possible modelling situations, defined with the System Modeling Language, that can lead to inconsistencies and studies the causes of them, more specifically if they are caused by the language specification or by how the specification is implemented in modelling tools. Moreover, we provide automatic validation checks to identify those inconsistencies by means of constraints defined with the Epsilon Validation Language. The results of this thesis are useful for modellers since it helps them in automatically identifying inconsistencies in models and in that way benefitting the most from modelling activities.
Table of Contents

1 Introduction 1
  1.1 Problem Formulation 2

2 Background 4
  2.1 Model-Based Systems Engineering 4
  2.2 Models and Metamodels 4
  2.3 System Modelling Language and Eclipse Papyrus for SysML 6
    2.3.1 Block Definition Diagrams 7
    2.3.2 Internal Block Diagrams 8
    2.3.3 Requirement Diagrams 8
    2.3.4 Eclipse Papyrus 10
  2.4 Epsilon Validation Language 11

3 Related Work 12

4 Method 13

5 Description of the work 14
  5.1 Modelling situations 14
    5.1.1 Ports with different types 14
    5.1.2 Item Flow with wrong data type 15
    5.1.3 Composition Loop 16
    5.1.4 Satisfy and Refine relationship not following the convention 18
  5.2 Implementation of the constraints 19
    5.2.1 Constraint 1 - Ports with different types 20
    5.2.2 Constraint 2 - Item Flow with wrong data type 21
    5.2.3 Constraint 3 - Composition Loop 23
    5.2.4 Constraint 4 - Satisfy relationship not following the convention 25
    5.2.5 Constraint 5 - Refine relationship not following the convention 25

6 Validation 27
  6.1 Constraint 1 and Constraint 2 27
  6.2 Constraint 3 28
  6.3 Constraint 4 and Constraint 5 29
  6.4 Results 30
  6.5 Threats to validity 30

7 Discussion 31

8 Conclusions 33
  8.1 Summary of the Work 33
  8.2 Further Development 33

References 34
1 Introduction

The importance of models has been growing over the last years: this is because they are used to describe and analyse a system under different points of view. On the other hand, the complexity of systems increased as well and, to deal with this complexity, models have been adopted to examine these systems under several aspects at different levels of abstraction: from the requirements elicitation to the test phase going through analysis and so on. In the context of Systems Engineering, the models are made by means of modelling languages (like the Unified Modelling Language and the System Modelling Language) [1]. During the development, these models are created in different phases: taking into account the development phases shown in figure 1, there are models all over these steps (Use Case diagrams in the Requirements Definition, Component diagrams in the System and Software Design and so on) [1].

Since models are used all over the realization of a project, they are crucial in the communication between the different stakeholders (people involved in the project). The requirements engineering can be taken as an example. It’s a process which aim is to find an answer to the following concerns: what problem should be solved, why such a problem needs to be solved and who should be involved [2]. Models are used in this phase to communicate with the other people involved in the project explaining and discussing the design proposals [1]. Models are useful to deal with the complexity of the projects because they represent, at different levels of abstractions, the systems the stakeholders work on. So, by means of the divide et impera paradigm, the system under examination is divided in sub-systems, which complexity is easier to handle.

The models are also used to automatically generate a part of the system: in fact, in the context of Model Driven Engineering, models (e.g. a UML class diagram or a SysML block diagram) can be used as input for tools that produce code in a specific programming language [3].

However, models with syntactical and semantic inconsistencies can lead to a potential spreading of faults and failures all over the development phases (because, in Model-Driven Engineering, models are commonly used in most of them) and among the different teams working on them. Moreover, it is possible that the wrongness in the models is not discovered in early phases: if this happens then all the problems are spread to other artefacts (generated code and so on), losing the Model-Driven approach benefits (it is cheaper and faster fixing problems appearing in the models instead the ones appearing in the artefacts like code).

Undetected model inconsistencies may arise when a language intentionally leaves a certain degree of freedom to the modeller or when the language implementation does not properly reflect its specification. Furthermore, the inconsistencies may arise because of the not properly defined syntax and semantics of domain-specific elements that are introduced by the modellers in the modelling language (thanks to its extension mechanism).

After this brief introduction, the problem concerning inconsistencies that can show up in models of a certain context will be presented in section 1.1. Then, all the information needed to have a
better understanding of the thesis will be shown to the reader in section 2, followed by the state of the art in section 3. After, how the problem has been dealt (section 4), with the description of the work carried on (section 5) and expected results. Section 6 shows the application of the implemented constraints to more general modelling situations to check their correctness. Finally, section 7 contains a brief discussion concerning the work of this thesis and section 8 embraces the conclusions and the possible future works.

1.1 Problem Formulation

Models that are not correct from a syntactic and a semantic point of view may bring negative consequences during the development phases of a project. In figure 2, there is a SysML internal block diagram made with Eclipse Papyrus (the most used tool in terms of Academic Research [4]). There is an inconsistency in the above-mentioned figure: the connected ports have different types (portA is a String while portB is an Integer). The two subsystems communicate through ports having different types: this may lead to critical errors because the two subsystems were expecting both an integer (or string) depending on the requirements. Instead, the communication points have not been modelled correctly (they have different types) rising problems during other phases of the project.

![Figure 2: SysML internal block diagram](image)

Models consistency, with respect to their syntax and semantics, is crucial during the whole development process. If this consistency is not respected (as it happens in figure 2), then there can be problems in the interaction between the results of teams working on the same project but focusing on different aspects. The artefacts (codes and so on) concerning these different aspects may have to interact with each other so, before the actual realization, the models are used. These models must not present ambiguities because of their language or the tools used to make them, otherwise, the teams will require more iterations to understand which models fit better the requirements. Inconsistencies like the one shown in figure 2 may be caused by several reasons. The inaccuracy of the chosen modelling language’s metamodel is one of these reasons: the authors of the metamodel may have not defined a correct syntax for their metamodel, allowing the existence of erroneous models. Moreover, the causes of the inconsistencies may be found in the used modelling tool: the implementation of the modelling language metamodel may have been done wrong in the tool. So, because of the inaccurate integration of the language in the tool, erroneous models may be defined.

When code is automatically generated from erroneous models like the one shown in figure 2 there will be faults that may lead to critical failures. As a matter of fact, taking into account the situation shown in figure 2, the different type of the ports is a fault: as already said before, the execution of one of the subsystems expecting a certain type of data but receiving another one will be faulty and, if not handled properly, it will result in a failure.

Faults and failures that may be discovered during the testing phase, making the engineers work again on earlier phases (design, etc.) to understand why the generated code does not respect the requirements. Furthermore, faults and failure may not be discovered during the testing phase, so they will exist also in deployment and execution phases where, in the case of safety-critical systems...
Constraints for avoiding SysML model inconsistencies

for example (an airplane, a nuclear plant and so on), they may lead to loss of human life and damages to the surrounding environment.

The automatic generation of code is not the only problem: code can be also manually generated from the models. In both the situations, the final implementation will have faults because of the models wrongness. So, the engineers are forced to carry out additional development iterations. For instance, additional iterations in the design phase are needed to ensure the correctness of the models. These iterations require a cost in terms of money and time.

A mechanism used to complement the modelling languages avoids this kind of problems: it can be implemented by means of rules defined through constraints to add to SysML. These constraints are defined by means of languages like the Epsilon Validation Language. Thanks to this mechanism, users can define new rules on the modelling languages (SysML and so on) refining their syntax to express information that may not be defined only with the models.

Research Goals and Research Questions

The main research goal of this thesis is to define a set of modelling situations that present inconsistencies like the one shown in figure 2 and a set of constraints that are useful to avoid these modelling situations.

- **RQ1**: What are the modelling situations in SysML that need to be constrained?
  - **RQ1a**: Are these modelling situations caused by how the language is specified or how it is implemented in modelling tools like Eclipse Papyrus?

- **RQ2**: How can constraints be defined at tool level in order to discover the identified modelling cases?

To achieve the research goal, an answer must be provided to the above mentioned research questions. First of all, the modelling situations in SysML have to be found: they will define the result set of the research. Then, it’s necessary to provide an answer to RQ1a because it will change what the constraints will be defined on. Finally, the answer of RQ2 will be a set of the constraints for the modelling situations found at the beginning: this set will complete the research goal.
2 Background

2.1 Model-Based Systems Engineering

According to [5], “a system consists of a set of elements that interact with one another, and can be viewed as a whole that interacts with its external environment”. An aeroplane is a system: the mechanical elements, that belong to the mechanical sub-system, interact among them and with the electronic elements, belonging to the electronic sub-system. A car is a system as well. Even the human body can be seen as a system.

During the last years, the complexity of systems kept growing, resulting in more demands to decrease the resources spent in favour of an increased capability of the product [5]. Nowadays, these systems have to be considered as components that can interact with each other, creating the concept of System of Systems (SoS) [5] (concept that increases more the complexity of the systems). In the context of Systems Engineering, an approach has been proposed: Model-Based Systems Engineering. This kind of approach is based on models that support all the steps of the system development [5]. The MBSE approach is based on artefacts represented by system models and by secondary artefacts (like deliverables) that are automatically generated, being less expensive and faster than code-based and document-based approach [6].

There are several benefits of MBSE [5, 7]:

- The models take into account the whole engineering problem;
- There is consistency between the product and the requirements;
- More complete and verifiable requirements;
- System presented from multiple perspectives;

2.2 Models and Metamodels

Essentially, a model is an abstract representation that describes a system under different points of view something that exists or that has to be realised. Models are not used only in the context of System Engineering: for example, the Figure 3 can be seen as a model of the circulatory system of the human body (that can be seen as a system). So, something that exists (the human body) is described under a certain point of view (the circulatory system).

![Figure 3: Circulatory system model](image)

To create these models, modelling languages are used: in the context of Systems Engineering, SysML is mostly adopted. Rules are needed to create models with these modelling languages: these
Constraints for avoiding *SysML* model inconsistencies

rules are defined by means of *metamodels*. In figure 4, there is a representation of the abstraction levels for models and metamodels. The two arrows on the right state that the levels are instances of the upper layer: *M1* represents instances of *M2*, while the rules defined in *M2* are instances of *M3*. The rounded arrow states that the concepts in *M3* are self-defined so there is no need of other abstraction layers.

To have a better idea of what is a metamodel, the *UML* one will be taken into account: it’s a set of *metaclasses*, which instances are the elements that can be designed at level M1 (components, classes, relationships and so on) and how they can interact [8]. For example, in figure 5, there is the *metaclass* for the class object:

As can be seen in figure 5, the *metaclass* defines the *class* element: every instance of this metaclass will be a *class*. This element inherits from the *metaclasses* *EncapsulatedClassifier* and *BehavioredClassifier* and it is composed of *properties* (thanks to the composition with *Property*...)
constraints and so on. The entire metamodel is made by means of metaclasses as the ones shown in figure 5: in this way, rules for the M1 level are defined in the M2 level. This means that a modelling language metamodel (i.e. the rules defined in M2) represents the actual specification of that modelling language: the elements in M1 defined with a certain modelling language must be conformed to its metamodel.

UML and SysML provide an extension mechanism for their metamodel, called profiling: it is useful to introduce domain-specific elements. As a matter of fact, UML is a general-purpose modelling language while SysML is a domain-specific language: this means that the latter is mainly used in a specific context (Systems Engineering) while the former can be used in more generic contexts. Furthermore, SysML itself has been created by extending the UML metamodel by means of Stereotypes, the core elements of a profile: for example, the element Block in SysML is an extension of the metaclass Class of the UML metamodel [9] (figure 6).

![Figure 6: Block Stereotype extending the metaclass of UML [9]](image)

It is possible to extend the SysML with the same mechanism: Stereotypes defined by the user can extend the SysML metamodel to create domain-specific elements.

### 2.3 System Modelling Language and Eclipse Papyrus for SysML

In the context of System Engineering, a modelling language has been created as an extension of UML: the System Modelling Language (SysML) [5, 10]. Thanks to this language, the modellers have help in creating systems that can be modelled by means of other languages like UML [5].

In figure 7, it is shown how SysML extends UML by adding diagrams and using the ones in common.

Below, a set of all the new diagrams in SysML. In the figure 8, boxes have different colours:

- The yellow boxes represent the diagrams that are completely new in SysML:
  - Requirement Diagram;
  - Parametric Diagram.

- The green boxes represent the diagrams that have been obtained by modifying the diagrams in UML 2:
  - Activity Diagram: obtained by modifying the Activity Diagram in UML 2;
  - Block Definition Diagram;
  - Internal Block Diagram.
  The last two are obtained by modifying the Class Diagram in UML 2.

- The black boxes represent all the diagrams that are used in both UML 2 and SysML.
Figure 7: How SysML extends UML [10]

Figure 8: All the diagrams in SysML [5]

2.3.1 Block Definition Diagrams

One of the most important diagrams in SysML is the Block Definition Diagram in which is possible to define systems, components, connections and so on by means of Blocks. By doing so, it is possible to model the system from a structural point of view [6].

The figure 10 represents a simple Block Definition Diagram (BDD): there is a System block that is composed of two other blocks (ComponentA, ComponentB). As can be seen, blocks are very similar to UML Classes: attributes, operations and constraints can be specified in each block. The
attributes represent the internal structure of a block: they can be other blocks as well or simpler types and they compose the block in which they are defined. As a matter of fact, the attributes of the block System are ComponentA and ComponentB:

![Block Definition Diagram](image)

Figure 10: Block Definition Diagram

A block is the core element of SysML because, as already mentioned before, it may be used to define every entity in the current context [6].

2.3.2 Internal Block Diagrams

Once the blocks have been defined in the BDD, it is possible to define their internal structure using the yInternal Block Diagram (IBD). One example has already been shown in figure 2. For the sake of the readability, another example showing the internal structure of the pre-defined System block will be presented in figure 11. The block System has two Parts, that are instances of the blocks ComponentA and ComponentB that interact through two Ports that are access points used to communicate [5].

The internal block diagram and the two parts are represented as children of the block System in the model.

2.3.3 Requirement Diagrams

The other new diagram is the Requirement Diagram: this type of diagram is used to display how the text-based requirements are related to other model elements of the context [6]. They are essentially used for traceability: whenever a new block is modelled in the system, a relationship may be drawn from this new element to the requirement that addressed its creation [6].

An example of a requirement diagram can be seen in figure 13: the most general requirement is the one with id=RQ1. Then, the two requirements with id=RQ1.1 and id=RQ1.2 compose the first one. These two are satisfied by the blocks CPU and GPU by means of the relationship Satisfy.

The relationship Satisfy is used to connect elements of the models that satisfy a certain requirement: in the example represented in figure 13, the blocks CPU and GPU satisfy the requirements...
RQ1.1 and RQ1.2. This relationship has a Requirement element to the Supplier end (arrowhead end) and, by convention, a Block element to the Client end (tailor end). There are no constraints concerning what can appear to the Client end [6].

Another relationship used is the containment: it states that a certain requirement is contained in another one. In this case, the requirements RQ1.1 and RQ1.2 are contained in the requirement RQ1.

There is another requirement diagram in figure 14, showing the usage of a Refine relationship. This dependency is used when a more concrete element has to refine a more abstract element. The former is placed at the Client end of the relationship, while the latter at the Supplier end. Even though every kind of element can appear at the Client end, a use case is used by convention while the Supplier end must be a requirement element [6]. In this case, the use case reversalCheck refines the requirement safetyRequirement. In the figure 15 there is the diagram of reversalCheck: as the
reader can see, it represents a more concrete element compared to the requirement shown in the figure 14.

2.3.4 Eclipse Papyrus

There are several tools implementing SysML. The one used in this thesis is Eclipse Papyrus for SysML. This tool has been chosen since it is open source and, according to [4], it is the most used one for academic research. Papyrus is an open source Eclipse-based framework used in the context of MDE to design models, by means of different modelling languages (UML, SysML and so on), and to automatically generate code (C, C++, Java) [11].
second one is the Model Explorer and it represents the actual Model of the current project.

2.4 Epsilon Validation Language

The Epsilon Validation Language (EVL) was born to complement the modelling languages by defining and evaluate constraints on models made with an arbitrary metamodel [12]. Users can define EVL modules to write and then apply EVL constraints. The constraints are based on a syntax that is briefly described below [12]:

1. A Context specifies the element which instances will be evaluated by the constraints;
2. Constraints: are defined with a name and a body and they "capture critical errors that invalidate the models";
3. Critiques are defined with a name and a body (like the constraints) but they differ from them because they detect non-critical situations;
4. A Guard applies limits to the application of the constraints and critiques to optimize the evaluation.

By means of these expressions, EVL is used to define constraints on models: these are needed since modelling languages cannot provide some information on the models as it can be seen in the diagram represented in figure 17.

![Diagram](image)

Figure 17: Requirement diagram

Let’s imagine a scenario in which a team has to follow a naming convention imposed by other stakeholders in their current project. The naming convention states: "the name of each requirement in a requirement diagram must start with RQ". In figure 17, one of the element does not respect the constraint. To avoid this kind of problem it’s enough to apply an EVL constraint to the SysML metamodel:

```evl
context examples!Class{
  constraint namingConvention{
    check{
      if (self.getAppliedStereotype("SysML::Requirements::Requirement") <> null)
        if (self.name.substring(0,2) <> "RQ")
          return false;
    }
    return true;
  }
  message: self + " does not respect the naming convention"
}
```

Listing 1: Example of EVL constraint applied to the model in figure 17

In listing 1 there is an EVL constraint: it is based on a context, which name starts with examples (that is the namespace of the model) and ends with the Metaclass which instances are going to be checked. The constraint is named namingConvention. To get the requirement elements it’s enough to check if the stereotype requirement is applied on the class (SysML is based on the UML metamodel). If the name of the requirement does not start with RQ then there will be a constraint violation.
3 Related Work

There already exist works concerning modelling languages specification that may have problems, focusing on their metamodel. For example, in [13], the authors work on UML 2.0 statecharts that have semantic variation points: these points are ambiguities in the metamodel of UML, providing certain freedom concerning the interpretation of that model, resulting in a generation of code that may be inconsistent with what the modellers have in mind. However, in [13], the work carried out focuses on only one type of UML models, i.e. the statecharts diagrams: they express the semantic variation points of these diagrams by means of models in order to not hard-code these variation points in the modelling tools. Furthermore, these models contribute to the generation (through model transformation) of a target model containing all the semantic and implementation choice are clearly expressed. This target model is needed for a consistent use of model-checking, simulation and so on. This means that the authors are working only on a focused subset of the UML metamodel: this is a difference with respect to the work carried on in this thesis. Potentially, the results of the following thesis may concern different types of models (requirements diagrams, block diagrams and so on) and so they will influence more points of the SysML and UML metamodels.

Another difference is that the authors of [13] do not consider a particular tool at all (in this thesis Eclipse Papyrus is taken into account): they start from the fact that there is an ambiguous point in the UML metamodel regardless of the modelling tool. The problem concerning the type of the connected ports has been raised also in [14]. The authors state that there are no rules concerning the type compatibility of connectable elements that are at the ends of a connector, leaving unclear how to model well-formed structures.

In [15], the author works on semantic variation points concerning the connectors in the UML metamodel. The number of instances concerning the components and ports can be clearly defined by means of multiplicities [8, 15] while, concerning the connectors, it is not the same thing. As a matter of fact, "the UML metamodel leaves the rules for matching the multiplicities of connected instances components and ports as a semantic variation point" [15]. Knowing precisely how the instances of these connectors will be realised is crucial for certain activities like the analysis, the deployment and the code generation. To deal with this semantic variation point, the author proposes a solution based on connection patterns and semantic rules (five combinations of multiplicities with specific semantic rules), a profile for UML called Explicit Connection Patterns (provides a specific stereotype for each connection pattern together with OCL expressions for the validation of these stereotypes) and on a generation algorithm used to produce, from a component-based model, a new one that contains the unrolled parts and ports (according to their multiplicity) of the input model [15]. The main difference is that the focus of the above-mentioned work is on the multiplicity of the connectors while, in this thesis, the focus is on the type of the ends of the connectors in a model. Moreover, the authors validates the stereotypes defined in the Explicit Connection Patterns profile by the application of constraints made with the Object Constraint Language (OCL). This is another difference because in this thesis another language is used to define the constraints (even though they are defined on the metamodel level as well). The last difference is that the work carried on in [15] does not rely on a particular tool because it starts directly from the UML metamodel.

In [16], the authors propose a mechanism concerning the inconsistencies of different UML models: its aim is to identify and solve these inconsistencies by means of logic rules. These logic rules are defined on only a subset of the UML metamodel that covers Class Diagrams, Sequence Diagrams and State Diagrams. The main difference here is that the work carried on in [16] focuses directly on the inconsistencies. So, models with inconsistencies are created and then these problems are solved through the proposed approach. Instead, in this thesis, the constraints specified for the UML and SysML metamodel aim to avoid the definition of modelling situations that will lead inconsistencies to the later development phases.
4 Method

Several steps have been performed in this thesis:

1. Study of the literature
   
   In this step, a study of what has been done from other people will be conducted. This step is necessary to find any modelling situations that have already been analysed (with or without the needed constraints). To perform this step, several papers regarding the experience reports of the SysML usage in different contexts will be studied. Furthermore, also the technical documentation of SysML will be analysed.

2. Define a set of models to detect the problematic modelling situations
   
   Once that the literature has been studied, examples of models that may present ambiguities have to be defined: these models will be carried on from existing examples or will be created from scratch. These models will be analysed to find out what are the possible constraints to add to the metamodel. The output of this step is a set of modelling situations that need the constraints.

   The definition of this set starts from the literature: as the reader will see in 5, some of the modelling situations under analysis are from the literature (papers and so on). They have been taken into account since they have not been treated by the authors of the above-mentioned sources but just cited. The others come from personal experience deriving from the work on other projects. Concerning the ones coming from personal experience, they are situations made trying to define something that has inconsistencies in its semantics. For example, the situation shown in 5.1.2, is incorrect from the semantic point of view even though its syntax is correct (as it will be clarified later, the SysML metamodel) allows the existence of this kind of model.

3. Analysis of the above-mentioned set for the constraints definition and implementation
   
   The set had to be analysed to find the constraints: these have been defined and formalized through mathematical rigour. After that, the constraints have been implemented following the definition and then applied to the modelling situations. The implementation has been made through EVL.

4. Validation of the results
   
   Finally, the set of constraint needed to be validated. The constraints have been applied on variations of the modelling situations to check if they work as expected by the mathematical formulation.
5 Description of the work

The work performed during this thesis will be shown in this section together with the results. Each modelling situation that has been carried out will be presented to the reader with a motivation: why this situation needs constraints? So, every modelling situations shown to the reader in this section is a result for the RQ1. Furthermore, these situations are made with Eclipse Papyrus: this means that every model analysed in this section is a result for the RQ1a. Moreover, the analysis of the UML and SysML metamodel highlighted that these models may be caused by the language specification. So, the models cover entirely the RQ1a. Concerning the RQ2, each subsection will show the high-level and the formal definition of each constraint. However, for the sake of the readability, the actual implementation of the constraints made with EVL will be shown in section 5.2.

All the situations shown in the following section are parts of the results of this thesis: as already briefly described in section 4, the study of the literature and personal experience brought up to life this kind of models. Situations like the ones shown 5.1.1, 5.1.4 could be analysed because they have been cited in scientific papers or in books concerning SysML.

5.1 Modelling situations

5.1.1 Ports with different types

The first modelling situation taken into account has been already shown to the reader during the first pages of this thesis: in a block diagram/internal block diagram it’s possible to connect two ports with different types. Since the ports are interaction points through which data are exchanged, we expect that two connected ports have the same type: if they don’t have the same type then, when the system will be actually implemented, there will be an inconsistency because of these two ports. This inconsistency is actually a fault that may lead to critical failures. The following figure represents the same problem already shown in earlier chapters: a System block is composed of two SubSystems connected by two ports of different types.

![Figure 18: Ports with different types](image)

RQ1a

The situation carried out in this section can exist because of a lack in the language (SysML). As a matter of fact there is no constraint appearing in the technical documentation of SysML [9] and this problem is not highlighted either in [5] either in [6]. Moreover, since this is a situation made with elements that concern directly the UML metamodel (Ports and Connectors), the technical documentation [9] of the UML metamodel has been checked: no similar problems appear in this document and, consequently, there are no constraints concerning it. Finally, as already said in 3, this problem is carried out also in [14].
RQ2

Formal constraint: two connected ports, portA and portB, must have the same type:

\[ C = \{c_1, c_2, \ldots, c_n\} \text{ Set of connectors} \]
\[ \text{CES} = \{E_1, E_2, \ldots, E_n\} \text{ Set of ConnectorEnd Sets} \]
\[ E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,m}\} \in \text{CES}, \ i = 1 \ to \ n \text{ Set of ConnectorEnd} \]
\[ \text{CE}_i = \{ce_{i,1}, \ldots, ce_{i,m}\} \text{ Set of ConnectableElement} \]
\[ \text{TYPE} \text{ Set of types} \]
\[ \text{ends} : C \rightarrow \text{CES}, \text{ends}(c_i) = E_i, 1 \leq i \leq n \]
\[ \text{role} : E_i \rightarrow \text{CE}_i, \text{role}(e_{i,j}) = ce_{i,j} \]
\[ \text{isPort} : \text{CE}_i \rightarrow \{\text{True, False}\} \]
\[ \text{type} : \text{CE}_i \rightarrow \text{TYPE} \]

Everything that is needed to formalize the constraint has been defined above. Now, the constraint that is going to be built is defined on the ports:

\[ \forall c_i \in C, \forall (e_{i,j}, e_{i,k}) \in \text{ends}(c_i), \text{role}(e_{i,j}) = ce_{i,j}, \text{role}(e_{i,k}) = ce_{i,k}, \]
\[ \text{isPort}(ce_{i,j}) = \text{True and isPort}(ce_{i,k}) = \text{True} \implies \text{type}(ce_{i,j}) = \text{type}(ce_{i,k}) \]

For each connector, all the ends are analysed: the ends that are ports are compared with the other ports. If they all have the same type, then the constraint is respected. The actual implementation of the constraint can be seen in 5.2.1

5.1.2 Item Flow with wrong data type

Another situation found concerns the itemFlow relationship: two ports can be connected by this kind of relationship. When this happens, there can be two problems:

1. The ports have different types (and so there is the situation shown in 5.1.1);
2. The ports have the same type but the type of data "flowing" between them is different (figure 19).

![Figure 19: Wrong type of the itemFlow](image)

As can be seen in the above figure, the type of the data flowing between the two ports is an Integer, while the two ports have the String type.

This situation may bring confusion to the developer team since it is not so clear what kind of data the ports are going to exchange.

RQ1a

Similarly to the problem carried out in 5.1.1, the situation shown in this section may exist because of a lack in SysML ([5], [6] and [9] have been checked as well as in 5.1.1). There are elements (ports and itemflow) that rely directly on the UML metamodel so, as already shown in 5.1.1, [8] has been checked as well.
RQ2

Formal constraint: The items conveyed by an itemflow item must have the same type of the source and the target:

\[
\begin{align*}
\text{IFS} & = \{ if_1, \ldots , if_n \} \quad \text{Set of InformationFlows } \\
S_i & = \{ s_{i,1}, \ldots , s_{i,k} \} \quad \text{Set of Sources of } if_i \\
S & = \{ S_1, \ldots , S_m \} \quad \text{Set of Sources Sets} \\
T_i & = \{ t_{i,1}, \ldots , t_{i,k} \} \quad \text{Set of Targets of } if_i \\
T & = \{ T_1, \ldots , T_m \} \quad \text{Set of Targets Sets} \\
C_i & = \{ c_{i,1}, \ldots , c_{i,k} \} \quad \text{Set of Conveyed of } if_i \\
C & = \{ C_1, \ldots , C_m \} \quad \text{Set of Conveyed Sets} \\
\text{TYPE} & \quad \text{Set of Types}
\end{align*}
\]

\[
\begin{align*}
\text{type} : S_i \cup T_i \cup C_i & \rightarrow \text{TYPE} \\
\text{getSources} : \text{IFS} & \rightarrow S \\
\text{getTargets} : \text{IFS} & \rightarrow T \\
\text{getConveyeds} : \text{IFS} & \rightarrow C
\end{align*}
\]

After defining everything needed, the constraint can be formalized as follows:

\[
\forall if_i \in \text{IFS}, \forall s_{i,j} \in \text{getSources}(if_i), \forall t_{i,k} \in \text{getTargets}(if_i), \text{type}(s_{i,j}) = \text{type}(t_{i,k}) \quad (1)
\]

In 1, all the sources and all the targets must have the same type. Then, the conveyed items are checked:

\[
\forall if_i, \forall c_{i,j} \in \text{getConveyeds}(if_i), \text{type}(c_{i,j}) = \text{type}(s_{i,1}), \quad s_{i,1} \in \text{getSources}(if_i) \quad (2)
\]

It's enough to compare the type of all the conveyed item with the type of just one source since sources and targets have the same type in this phase. The actual implementation of the constraint can be found in 5.2.2.

5.1.3 Composition Loop

Given two blocks, A and B, A appears as a property of B and vice-versa. This happens when there is a loop composition (figure 20).

![Figure 20: Loop composition](image)

However, a more general situation can be highlighted, like the one shown in figure 21 (not only with two blocks but with n blocks) This kind of situation may create no-sense models in a project: for example, since hardware can be modelled with blocks, there can be a situation in which a block car is composed of a block wheels and, at the same time, the latter is composed of a car. The context around car and wheels is known enough to avoid this situation but, if the context concerns a specific domain that may not be so known, then problems like this may arise.

RQ1a

Again, the situation shown in this section may exist because of a lack in SysML ([5],[6] and [9] have been checked as well as in 5.1.1). There are elements (ports and itemflow) that rely directly on the UML metamodel so, as already shown in 5.1.1 and 5.1.2, [8] has been checked as well.
RQ2

*Formal constraint:* no composition loops should appear in a Block Definition Diagram. A Block Definition Diagram may be seen as a directed graph $G = (V, E)$ in which the nodes are the blocks and the edges represent the composition relationship between two blocks.

$$V = \{Bl_1, \ldots, Bl_n\}$$

The set of attributes of the block $Bl_i$ is defined as follows:

- $Attr_i = \{attr_{i,1}, \ldots, attr_{i,k}\}$ Set of Attributes
- $A = \{Attr_1, \ldots, Attr_n\}$ Set of Attributes Sets
- $attrs : V \rightarrow A$, $attrs(Bl_i) = Attr_i = \{attr_{i,1}, \ldots, attr_{i,k}\}$
  - TYPE Set of Types
  - $type : Attr_i \rightarrow TYPE$

The edges are defined as follows:

$$(Bl_i, Bl_j) \in E \iff \exists attr_{j,i} \in attrs(Bl_j) \mid type(attr_{j,i}) = Bl_i$$

So, $Bl_i$ composes $Bl_j$ (there is a direct edge from the former to the latter) if and only if the former appears as one of the attributes of the latter.

The corresponding directed graph of the model shown in figure 21 can be seen in figure 22.

The following function verifies if there is a path between two nodes in the graph:

$$path : V \times V \rightarrow \{True, False\}$$

$\text{path}(\text{Block37, Block38}) = True$, Block37 and Block38 are defined in figure 21. Finally, the constraint can be formalized as follows:

$$\forall Bl_i, \exists Bl_j \in V, Bl_i \neq Bl_j, \text{path}(Bl_i, Bl_j) = True \implies \text{path}(Bl_j, Bl_i) = False$$

That is, if there is a path from a node to another, the vice versa shall not be possible.
5.1.4 Satisfy and Refine relationship not following the convention

- In the Requirements Diagram, the satisfy relationship can be used without following the convention. The convention states that there should be a block at the client end but, as it can be seen in the figure 23, this convention may not be followed;

- In the Requirements Diagram, the refine relationship can be used without following any convention: there are no constraints concerning what can appear at both the end of the relationship (while the convention states that at the supplier end there should be a requirement and at the client end there should be a UseCase).

Both of the two problems can be seen in figure 23. The convention has been carried out from [6]

These problems create inconsistencies in the model: it may not be possible to know what really satisfies a requirement (so it’s not sure if it is satisfied by something that exists or that has to be realized/modelled) or what refines a requirement (so it is not sure if all requirements have been elicited in the right way).
RQ1a
No constraint appears in the SysML technical documentation [9]: however, in [6], it is stated that there is a convention that should be followed concerning the relationships under examination. Since it is only a convention, the engineers are not obliged to follow it, and they can make models like the one shown in figure 23.

RQ2

**Formal constraint (Satisfy relationship):** the client end of each Satisfy relationship should be a Block element.

\[
\begin{align*}
\mathcal{SAT} &= \{sat_1, ..., sat_n\} \text{ Set of Satisfy objects} \\
C_i &= \{cl_{i,1}, ..., cl_{i,m}\} \text{ Clients of Satisfy } sat_i \\
\mathcal{CLS} &= \{C_1, ..., C_n\} \text{ Set of Clients Sets} \\
getClients : \mathcal{SAT} &\rightarrow \mathcal{CLS}, \text{ getClients}(sat_i) = C_i \\
isABlock : C_i &\rightarrow \{True, False\}
\end{align*}
\]

The constraint can be formalized as follows:

\[\forall sat_i \in \mathcal{SAT}, \forall cl_{i,j} \in \text{getClients}(sat_i), \text{isABlock}(cl_{i,j}) = True\]

All the elements at the client end of a Satisfy relationship have to be Blocks.

**Formal constraint (Refine relationship):** the client end of each Refine relationship should be a UseCase element

\[
\begin{align*}
\mathcal{REF} &= \{ref_1, ..., ref_n\} \text{ Set of Refine objects} \\
C_i &= \{cl_{i,1}, ..., cl_{i,m}\} \text{ Clients of Refine } ref_i \\
\mathcal{CLS} &= \{C_1, ..., C_n\} \text{ Set of Clients Sets} \\
getClients : \mathcal{REF} &\rightarrow \mathcal{CLS}, \text{ getClients}(ref_i) = C_i \\
isAUseCase : C_i &\rightarrow \{True, False\}
\end{align*}
\]

The constraint can be formalized as follows:

\[\forall ref_i \in \mathcal{REF}, \forall cl_{i,j} \in \text{getClients}(sat_i), \text{isAUseCase}(cl_{i,j}) = True\]

5.2 Implementation of the constraints

In this subsection, the actual implementation of the constraints will be shown. This section will be divided for each code, referring to a particular situation found in the previous section. Furthermore, a diagram showing the evaluation order of the constraints of each listing will be shown. The syntax of these diagrams is very simple:

- The rectangles contain the name of the constraint that is evaluated;
- The arrows represent the path depending on the result of the evaluated constraint: there are two possibilities for each constraint, True or False;
- The ellipses represent what is shown to the user;
- The diamond represents an And operator;
- The circles represent the start/end of the flow.

Concerning the listings 4 to 6, they will not have the corresponding diagrams since they are based on operations: functions called when needed.
5.2.1 Constraint 1 - Ports with different types

```java
context thesis!Connector{
  constraint differentType{
    guard: self.satisfies("noOneNull")
    check: self.end.role ->forall(e | e.type = (self.end.role ->select(e | e<>null).first().type))
    message: "The connector " + self.name + " has ends with different types"
  }
  constraint noOneNull{
    check: self.end.role ->forall(e | e <> null)
    message: "Some element is null"
  }
  constraint noTypeNull{
    check: self.end.role ->forall(e | e.type <> null)
    message: self.name + " has null ends"
  }
}
```

Listing 2: Code Constraint 1

![Flow Diagram for listing 2](image)

The implementation of the Constraint 1 is represented in listing 2: the context is represented by the `connector metaclass`. For each instance of the `connector` in the model, the constraint checks the types of the ends: if they are all the same, then the constraints validates successfully the model. Before this evaluation, the ends and their types are checked: if they are null then the flow stops.
5.2.2 Constraint 2 - Item Flow with wrong data type

```plaintext
context thesis!InformationFlow{

constraint conveyedTypeMatch{
  guard: self.satisfies("typeSourceAndTargetEquals")
  check{
    var conv = self.conveyed;
    for(c in conv){
      if(c.name <> self.source.first.type.name)
        return false;
    }
    return true;
  }
  message{
    var conveyeds = self.conveyed;
    var types = "";
    for(conv in conveyeds){
      types+=conv.name + ", " ;
    }
    types=types.substring(0, types.length()-1);
    if(conveyeds.size()==1){
      return "The type of sources and targets of " + self.Name + 
      " is " + self.source.first.type.name + " and " + 
      self.target.first.type.name + 
      ". The type of the conveyed item is: " + types;
    }
    return "The type of sources and targets of " + self.Name + 
    " is " + self.source.first.type.name + 
    ". The types of the conveyed item are: " + types;
  }
}

constraint typeSourceAndTargetEquals{
  guard: self.satisfies("sourcesEqualTypes")
  and self.satisfies("targetsEqualTypes")
  and self.satisfies("conveyedTypeEqual")
  check{
    var sources = self.source;
    var targets = self.target;
    for(s in sources){
      for(t in targets){
        if(s.type.name <> t.type.name)
          return false;
      }
    }
    return true;
  }
  message{
    return self.Name + " has sources and targets of different types";
  }
}

constraint sourcesEqualTypes{
  guard: self.satisfies("typesSourcesNotNull")
  check{
    var sources = self.source;
    var fSource;
    for(s in sources){
      if(s <> null){
        fSource = s;
        break;
      }
    }
    for(s in sources){
      if(s <> null){
        if(s.type.name <> fSource.type.name)
          return false;
      }
    }
}
```

21
} 
\} 
return true; 
message: self.Name + " has sources with different types"
} 
\} 
constraint targetsEqualTypes{ 
guard: self.satisfies("typesTargetsNotNull") 
check{ 
var targets = self.target; 
var fTarget; 
for(t in targets){ 
if(t<>null){ 
fTarget = t; 
break; 
} 
} 
for(t in targets){ 
if(t.type.name <> fTarget.type.name){ 
return false; 
} 
} 
return true; 
} 
message: self.Name + " has targets with different types"
} 
\} 
constraint conveyedTypeEqual{ 
check{ 
var cvyds = self.conveyed; 
var fCvy; 
for(c in cvyds){ 
if(c <> null){ 
fCvy = c.name; 
break; 
} 
} 
for(c in cvyds){ 
if(c.name <> fCvy){ 
return false; 
} 
} 
return true; 
} 
message: self.name + " has conveyed with different types"
} 
constraint typesSourcesNotNull{ 
check: self.source -> forall(s | s.type <> null) 
message: self.name +" has sources with null type"
} 
\} 
constraint typesTargetsNotNull{ 
check: self.target -> forall(t | t.type <> null) 
message: self.name +" has targets with null type"
} 
\} 
Listing 3: Code Constraint 2
In listing 3, a lot of conditions have to be checked in order to validate successfully the model. For each InformationFlow, the types of the sources and the targets (sourceEqualTypes constraint and targetsEqualTypes constraint) will be checked. Furthermore, the type of the conveyed items must be checked (conveyedTypeEqual): to avoid inconsistencies, all the conveyed items must have the same type. Finally, the type of the conveyed items must be compared with the sources and targets type (conveyedTypeMatch constraint). The diagram shown in figure 25 requires particular attention: the constraint typeSourceAndTargetEqual may be evaluated only after the evaluation of the other three constraints: sourcesEqualTypes, targetsEqualTypes, conveyedEqualTypes. sourcesEqualTypes and targetsEqualTypes may be evaluated only after the evaluation of typesSourcesNotNull and typesTargetsNotNull respectively. These two constraints are evaluated together with conveyedEqualTypes. These constraints are evaluated together, that’s why they are at the same level in the diagram (conveyedEqualTypes is not at the same level of typesSourcesNotNull and typesTargetsNotNull for the sake of readability). The AND operator evaluates the combination of the three constraints evaluation results: it’s True if and only if the evaluation of all the three constraints is True. Otherwise, it’s False and one (or more) of the messages in the ellipses will be shown.

5.2.3 Constraint 3 - Composition Loop

```plaintext
constraint noLoop {
```
check {
  var allBlocks = getAllBlocks();
  var results: Sequence<Class>;
  for (bl in allBlocks) {
    var visited: Sequence<Class>;
    visited.clear();
    var neighbours = getNeighbours(bl);
    if (neighbours.size() > 0) {
      if (dfs(bl, neighbours.at(neighbours.size() - 1), visited, neighbours)) {
        results.add(bl);
      }
    }
  }
  return results.size() == 0;
}
message {
  var name = "";
  for (bl in results) {
    name += bl.name + " , ";
  }
  name = name.substring(0, name.length() - 2);
  name += " are part of one or more loops";
  return name;
}
operation dfs(source: Class, current: Class, visited: Sequence<Class>, neighbours: Sequence<Class>): Boolean {
  if (neighbours.size() == 0) {
    return false;
  }
  if (checkAttributes(current, source)) {
    return true;
  }
  for (neighCurr in getNeighbours(current)) {
    if (not visited.contains(neighCurr)) {
      neighbours.add(neighCurr);
    }
  }
  visited.add(current);
  neighbours.remove(current);
  return dfs(source, neighbours.last(), visited, neighbours);
}
operation checkAttributes(current: Class, source: Class): Boolean {
  var sourceOwnAttributes = source.ownedAttribute;
  for (sAttrs in sourceOwnAttributes) {
    if (sAttrs.type <> null) {
      if (sAttrs.type.name == current.Name) {
        return true;
      }
    }
  }
  return false;
}
operation getNeighbours(bl: Class) {
  var neighbours: Sequence<Class>;
  var allBlocks = getAllBlocks();
  for (cl in allBlocks) {
    if (bl <> cl) {
      if (checkAttributes(bl, cl)) {
        neighbours.add(cl);
      }
    }
  }
  return neighbours;
}
operation getAllBlocks() {
  var allBlocks: Sequence<Class>;
  for (cl in Class.allInstances()) {
    if (cl.getAppliedStereotype("SysML::Blocks::Block") <> null) {
      allBlocks.add(cl);
    }
  }
}
According to the formulation in 5.1.3, a Block Diagram can be seen as a directed graph. So, to check the presence of any loop, a Depth First Search based algorithm has been implemented. The reader can see this implementation in the listing 4. There are some things that need particular attention: there is no context and the use of the method `getAppliedStereotype` (operation `getAllBlocks()`). There is no context since the blocks will be examined all together and not one by one. The method `getAppliedStereotype` is needed to take the Blocks elements from the Class objects since the metamodel under analysis is the UML one. All the instances of the metaclass `Class` are taken and parsed: if the method `getAppliedStereotype` with argument the name-space of the Block stereotype is not null then that class is a block and it will be added into a list. The DFS based algorithm will be applied to the elements of that list: each block that is part of a loop will be highlighted to the user.

5.2.4 Constraint 4 - Satisfy relationship not following the convention

Listing 5: Code Constraint 4

critique satisfies{
    check{
        var abstractions = Abstraction.allInstances();
        var satisfies: Sequence<Abstraction>;
        var noRespecting: Sequence<Abstraction>;
        for (ab in abstractions) {
            if(ab.getAppliedStereotype("SysML::Requirements::Satisfy") <> null){
                satisfies.add(ab);
            }
        }
        for (s in satisfies) {
            for (c in s.client) {
                if (c.getAppliedStereotype("SysML::Blocks::Block") == null) {
                    noRespecting.add(s);
                }
            }
        }
        return noRespecting.size() == 0;
    }

    message{
        var names = "";
        for (nr in noRespecting) {
            names += nr.Name + ", ";
        }
        names = names.substring(0, names.length() - 2);
        if (noRespecting.size() == 1) {
            return names + " has at least one client that is not a Block";
        } else {
            return names + " have at least one client that is not a Block";
        }
    }
}

5.2.5 Constraint 5 - Refine relationship not following the convention

Listing 5: Code Constraint 4

critique refines{
    check{
        var abstractions = Abstraction.allInstances();
        var refines: Sequence<Abstraction>;
        var noRespecting: Sequence<Abstraction>;
        for (ab in abstractions) {
            if (ab.getAppliedStereotype("SysML::Blocks::Refine") <> null) {
                refines.add(ab);
            }
        }
        for (r in refines) {
            if (r.getAppliedStereotype("SysML::Blocks::Block") == null) {
                noRespecting.add(r);
            }
        }
        return noRespecting.size() == 0;
    }

    message{
        var names = "";
        for (nr in noRespecting) {
            names += nr.Name + ", ";
        }
        names = names.substring(0, names.length() - 2);
        if (noRespecting.size() == 1) {
            return names + " has at least one client that is not a Block";
        } else {
            return names + " have at least one client that is not a Block";
        }
    }
}
if (ab.getAppliedStereotype("SysML::Requirements::Refine") <> null)
    { refin.es.add(ab); }

    for (s in refin.es)
        { for (clis in s.client)
            { if (clis.type.name <> 'UseCase')
                noRespecting.add(s); }

        for (suppl in s.supplier)
            { if (suppl.getAppliedStereotype("SysML::Requirements::Requirement") == null
                and not noRespecting.contains(s))
                noRespecting.add(s); }

        return noRespecting.size() == 0;
        }
    }

message{
    var names = "";
    for (nr in noRespecting)
        { names += nr.Name + " , ";
    }

    names = names.substring(0, names.length() - 2);
    if (noRespecting.size() == 1)
        return names + " does not respect the convention";
    return names + " do not respect the convention";
}

Listing 6: Code Constraint 5

The codes shown in listings 5 and 6 are very similar: both have no context since they are based on all instances of the Abstraction relationship. Then, the method getAppliedStereotype is applied to these instances to find the refine and satisfy relationships. Then, according to the mathematical formulation shown in 5.1.4, the suppliers/clients are checked.
6 Validation

The defined and implemented constraints are applied to other similar modelling situations in order to check if they work. There will be a subsection for each constraint, in which the new models are presented and the result of the constraint’s application will be shown.

6.1 Constraint 1 and Constraint 2

Let’s consider the following model (figure 26): the block PortType is used to give a more complex type to some of the ports in the model while the block Data gives a more complex type for the data conveyed by the ItemFlows. Now, let’s consider the internal structure of the block ComponentA (figure 27): the Constraint 1 and Constraint 2 will be applied to it.

Figure 26: A new model taken into account

Figure 27: Internal structure of ComponentA
In figure 28, there is a screen representing the validation tab of Eclipse Papyrus, showing the result of the constraints’ application. As it can be seen, the Constraint 1 detects errors on four Connectors: Connector1, Connector3, Connector4 and Connector5 which ends have different types. Connector2 has ends with the same type, so the constraint is respected by it. Constraint 2 detects three errors on ItemFlow8, ItemFlow9 and ItemFlow11: source and target of ItemFlow9 and ItemFlow11 have different types while the ItemFlow8 has Conveyed Items which types (String, Integer) are not coherent with the source and target types. The constraints work as expected from the high-level and mathematical formulation.

6.2 Constraint 3

Figure 29 represents the new model the constraint 3 is applied to. There are seven blocks connected between each other by means of the composition relationship. For the sake of readability, in figure 30 is represented the corresponding directed graph of the new model. It has been made to easily show to the reader the loops in the model and the blocks that are part of these loops. The figure 31 represents the screen of the validation tab taken from Eclipse Papyrus after the application of Constraint 3 to the model shown in figure 29. Taking into account the graph shown in figure 30, BlockA, BlockD, BlockE, BlockF and BlockG are part of loops: in particular, BlockA
and BlockG are part of two loops (BlockA-BlockG-BlockF and BlockA-BlockG-BlockD-BlockE). BlockC has no outgoing edges so it does not appear as an attribute of other blocks. BlockB has no incoming edges but has two outgoing edges: to BlockD and to BlockA so it will appear as an attribute but no other blocks will be a property of this block. As a matter of fact, BlockB and BlockC cannot be part of any loops and so the constraint does not take them into account. The Constraint works as expected by the high-level and mathematical formulation.

6.3 Constraint 4 and Constraint 5

In the figure 32 there is a Requirement Diagram: the constraints 4 and 5 will be applied to the model expressed in this diagram. The results of the constraints’ application is shown in figure 33.
through a screen taken from the validation tab of *Eclipse Papyrus*. As can be seen, the relationships

![Validation](image)

Figure 33: Results of the constraints 4 and 5 application

*Refine8, Refine11, Refine13* do not respect the constraint (the clients are not UseCases) while *Refine17* does not trigger the warning because its client is actually a UseCase. So, the Constraint 5 works as expected.

There are three *Satisfy* relationships that trigger the Constraint 4: *Satisfy7, Satisfy9, Satisfy10*. Their client is not a block while the client of *Satisfy14* is a Block and does not trigger the warning. So, the Constraint 4 works as expected.

### 6.4 Results

As already briefly discussed in each subsection mentioned above, all the constraints work as expected in these new models. The modelling situations shown in the previous subsection are different from the ones taken as examples in section 5. Their complexity is higher and they are more general: for example, taking into account the model shown in section 6.1, it can be seen that the relationships associate ports belonging to the enclosing components to ports belonging to subcomponents (in the model shown in section 5.1.1 the association is only between two ports belonging to subcomponents). More general models help in the validation of the applied constraints.

### 6.5 Threats to validity

Even though *Eclipse Papyrus* was the chosen modelling tool in this thesis, another one has been used to understand if *Eclipse Papyrus* is the only tool that does not provide support when it comes to detect the inconsistencies carried out in this thesis. The above-mentioned modelling situations have been defined by means of another open source modelling tool: *Modelio* [17]. However, this modelling tool had the same lack of support concerning the detection of these modelling situations.

Detection tests have been performed also with another commercial tool: *MagicDraw* [18]. However, not all the modelling situations have been tested with this tool due to lack of time and due to the priority given to *Modelio* (because it is open-source and free like *Eclipse Papyrus*). As a matter of fact, by means of *MagicDraw*, only the modelling situations that have been shown in sections 5.1.1 and 5.1.3 have been defined in this tool and the results were the same of *Eclipse Papyrus* and *Modelio*: no support concerning the detection of the modelling situations.
7 Discussion

The work carried on in this thesis will be useful for the engineers during the definition of their models. The research goal proposed in section 1.1 has been fully accomplished in the boundaries defined through the thesis. As a matter of fact, an answer to the RQ1 has been provided but, the modelling situations that have been found (and consequently the EVL constraints), may not cover all the potentially lacks of the SysML metamodel. This claim can be justified by the fact that the thesis project has been carried on for a certain period of time: not all the time could be used to search the potential modelling situations so, saying that the language has been entirely covered cannot be taken for granted. It is possible to claim that the answer provided to RQ1 may be extended as new erroneous modelling situations can be found thanks to a deeper study of the literature and to the growing experience while working on other projects based on SysML.

Concerning the RQ1a, it was unexpected finding out that all the modelling situations carried on in this work can exist because of the language specification: this means that the metamodel taken into account has not been covered properly in some of its parts. This means also that the modelling tool does not provide further support to the language specification: no additional constraints are provided by the tool itself. Even though Eclipse Papyrus was the chosen tool to carry on this work, other modelling tools have been tried to understand if it was possible to make the modelling situations defined in section 5. The results were the same of the ones had with Eclipse Papyrus because, as already said in section 5, these modelling situations can exist because of how the SysML metamodel has been defined. So, the results will be the same despite the used modelling tool.

Concerning the RQ2, EVL has been chosen for the constraints implementation. However, this language was not the only option taken into account. The Object Constraint Language is a valid alternative to EVL: as already mentioned before, the former was the chosen one for the implementation of the constraints. However, due to technical problems, OCL files were not working in Eclipse Papyrus so EVL has been chosen, achieving the expected results. Anyhow, the same constraints defined in EVL can be implemented in OCL as well. All the constraints defined in section 5 have been implemented without particular difficulties except for the constraint shown in section 5.1.3: EVL does not provide a mechanism to create or handle graphs like other languages do (in Java, for example, Key-Value structures can be used). As it can be seen in listing 4, a proper operation has been defined to get all the neighbours of a block: to get these neighbours, all the attributes of all the blocks above mentioned have to be checked by means of the operation checkAttributes(current: Class, source: Class): this operation checks if current appears as a property of source. These two operations are repeated several times during the execution of the algorithm: this means that, applying this constraint on models which complexity regarding their size is very high, the execution may require a lot of time.

Regarding the version of SysML, the work of this thesis has been based on the version 1.4 of the SysML specification. However, the convention stated in [6] regarding the modelling situation shown in section 5.1.4 has been risen from the version 1.3 of the specification. Moreover, from the SysML specification (version 1.3) it is clear that the modelling situations defined by means of the version 1.4 could have been defined with also that version. This prove how the SysML specification authors gave a certain degree of freedom also from initial versions of SysML.

Another interesting point of discussion is the severity level of the constraints: they can be a warning or an error. While the constraints 5.1.1, 5.1.2 and 5.1.3 will show an error, the remaining ones will show a warning if the validation goes wrong. This distinction has been made because the last two constraints impose the users to follow the convention stated in [6]. However, since it is a constraint regarding a convention, the severity level has been relaxed to a warning. As the reader can see in the codes shown in section 5.2, the difference between error and warning is risen up by the use of the keywords constraint and critique: the first one is used to detect errors that may be critical in the context of the project in which these errors appear while the second one is used to detect those modelling situations that does not necessarily bring to fatal errors and so they may be ignored.

Finally, it’s interesting to discuss about the SysML specification, which was thought to giving
a high degree of freedom to the modeller. Freedom is meant as the possibility for the modeller to define erroneous models, like the ones shown in this thesis. However, providing this freedom without proper automated validation checks, as the ones proposed in this work, may lead to undetected errors in models, which can propagate to the final product. The definition of EVL constraints in the SysML implementation can help the modeller in identifying possible issues once the models are ready to be validated.
8 Conclusions

8.1 Summary of the Work
The work shown in this thesis regards a modelling tool, Eclipse Papyrus, that is based on modelling languages like UML and SysML. It has been shown that Eclipse Papyrus allows the creation of modelling situations that have inconsistencies in the syntax and the semantics of the used language. Furthermore, it has been found out that it is not only a problem of the tool, but there are some lacks in the SysML metamodel.

Constraints are used to solve these problems: they have been first defined using a high-level language and formalized through mathematical rigour, then they have been implemented using a proper language. In the beginning, the Object Constraint Language was planned to be used but, since there have been problems applying OCL files (unexpected crashes, models not correctly parsed by the OCL code and so on) so, the Epsilon Validation Language has been used. Through the latter, the formalised constraints have been translated into code and the applied successfully on the models that needed them.

The application succeeded without problems: all the elements that needed the constraints have been detected. In Section 6 there are screen-shots of Eclipse Papyrus in which the results of the constraints are proved.

The contributions of this work born to help the engineers while modelling with SysML: the need for this contribution arises from the fact that having modelling situations that are not consistent with respect to the requirements of a project implies, in a company, for example, a loss of resources in terms of money and time since the teams may require re-iterations on models.

8.2 Further Development
The work carried out in this thesis regards only a subset of the possible modelling situations made with SysML. That means: other models that create inconsistencies in a project may exist so other research can be lead on them and other constraints may be defined and implemented. Furthermore, it would be interesting studying other commercial modelling tools together with Eclipse Papyrus to check if the possible new modelling situations are a result of a lack in the metamodel or the result of a lack in the tools taken into account.
References


