A Methodological Framework for SysML: a Problem Frames-based Approach

Pietro Colombo Vieri Del Bianco Luigi Lavazza Alberto Coen-Porisini
Dipartimento di Informatica e Comunicazione
Università dell'Insubria
Via Mazzini 5 - 21100 Varese (Italy)
{pietro.colombo | vieri.debianco | luigi.lavazza | alberto.coenporisini}@uninsubria.it

Abstract

Recently, SysML has been adopted by the Object Management Group as a modelling language for Systems Engineering. SysML is a UML profile that represents a subset of UML 2 with extensions. A wide adoption of the language could be hindered by the lack of a methodology that drives the modelling activities.

Problem Frames (PFs) are a rigorous approach to requirements modelling that has the potential to improve the software development process. Unfortunately, PFs are not supported by an intuitive notation and easy to use tools. As a consequence, their adoption in industry is limited.

This paper explores the possibility of exploiting the PFs ideas in the context of SysML models. The goal is to provide model-based development processes using SysML with a set of concepts and guidelines that are sound and have already been used and validated.

1. Introduction

Model-based development is increasingly gaining popularity. There is a continuously growing number of organizations that adopt model-based development processes. In this scenario UML [17, 16] is the most widely adopted modelling language. Although UML has had the merit of providing an easy to use and reasonably expressive language supporting model based development, it has several limitations that have been widely discussed in the literature. SysML is a UML profile that extends a subset of UML 2 and recently it has been officially adopted by the OMG as a modelling language for Systems Engineering [15]. It remains largely compatible with UML, while it introduces modification of existing diagrams and totally new diagrams in order to overcome the limits of UML as a system modelling language.

However SysML is not provided with a specific modelling methodology. This lack is particularly relevant since SysML offers linguistic means to describe and allocate requirements, but does not provide users with any hint on how to define and structure requirements nor on how to classify the problems they represent.

On the contrary, Problem Frames (PFs) [8, 7] drive developers to understand and describe the problem to be solved. The PFs approach has the potential to dramatically improve the early lifecycle phases of software projects. Nevertheless, PFs have some limitations that hinder their application in industrial software development processes. In particular, they are not provided with adequate linguistic support. The modeller has to choose a suitable language to predicate about phenomena. Moreover, the PFs approach causes a sort of "impedance mismatch" with respect to the languages employed in the subsequent development phases.

Therefore, it appears convenient to integrate modelling languages with the PFs approach. In fact, in a previous work [12] the authors explored the integration of UML with the PFs concepts. The result was that UML-based development can be equipped with a sound and rigorous method for requirements analysis and representation. From the PFs point of view, the method is equipped with the linguistic support provided by UML that is reasonably expressive and easy to use. Nevertheless, the experimental integration of UML and PFs revealed that UML suffers from several limitations that hinder its usage in combination with PFs. In fact, UML is too much design-oriented, and does not support modelling at the correct level of abstraction (PFs generally require a level of abstraction higher than the one adopted by UML).

SysML promises to perform better than UML with respect to the aforementioned issues. Therefore, the purpose of this paper is to integrate PFs concepts with SysML and verify how well the linguistic constructs of SysML support PFs-based requirements modelling. Although some researchers have studied how to extend PFs based methods to

1530-1362/07 $25.00 © 2007 IEEE
DOI 10.1109/ASPEC.2007.56

the design phases, these topics are out of the scope of this paper. Here we address exclusively the requirements modelling phase.

The paper is organized as follows: Section 2 briefly describes Problem Frames and presents a running example used throughout the paper. Section 3 provides a short introduction to SysML and then discusses the integration of SysML and PFs with the help of the example. Section 4 accounts for related work. Finally, Section 5 summarizes the results and describes future developments.

2. Problem Frames

Problem Frames (PFs) were defined by M. Jackson [8, 7] as a rigorous approach to software requirements analysis. In what follows we provide a short introduction to the PFs approach; the interested reader may refer to the book on PFs ([8]) and/or to the many papers on the subject (e.g., `http://www.ferg.org/pfa/ index.html` for a more comprehensive description of PFs).

2.1. A short introduction to PFs

Problem Frames are based on the concept that user requirements are about relationships in the real world and not about functions that the software system must perform. Rather, the desired relationships in the real world are achieved with the help of a machine. However, in the requirements analysis phase, the Machine is only specified as far as its role in the real world is concerned, that is, only the interface between the machine and the problem domain needs to be specified, while the machine internals are left unspecified (they will be addressed in the design phase).

Thus, the first task is to understand the context in which the problem is set, and to represent it by means of a *context diagram*. The context diagram shows the various *problem domains* in the application domain along with their connections, and the Machine and its connections to (some of) the problem domains.

A domain is simply a part of the world that we are interested in. It consists of phenomena such as individuals, events, states, relationships, and behaviors. An interface is a place where domains overlap, so that the phenomena in the interface are shared, thus allowing connection and communication between domains. Usually one domain controls a set of shared phenomena, which are visible by other domains. The fact that a phenomenon is shared between two domains does not imply any kind of data flow or message sending since defining the policies that govern the interfaces is left to the design phase.

Problem diagrams add requirements to context diagrams. Requirements are attached to domains and specify conditions involving the phenomena of those domains (including the private, non-shared ones).

An interface that connects a problem domain to the Machine is called a *specification interface*. The goal of the analyst is to develop a specification of the behavior that the Machine must exhibit at its interface in order to satisfy the user requirements.

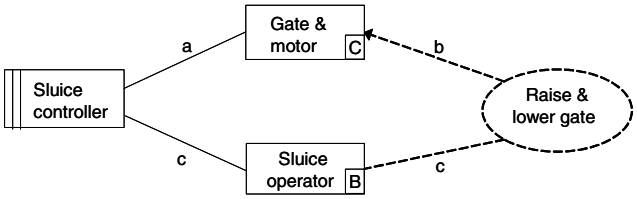
A PF is a description of a recognizable class of problems, and thus in some sense, problem frames are problem patterns. Jackson identified five problem frames [8], each of which having its own problem diagram.

2.2. The Sluice Gate example

In order to illustrate the applicability of PFs concepts in SysML modelling, the well known problem of controlling a sluice gate [8] is used as a running example.

A small sluice, with a rising and a falling gate, is used in a simple irrigation system. A computer system is needed to raise and lower the sluice gate in response to the commands of an operator. The gate is opened and closed by rotating vertical screws. The screws are driven by a small motor, which can be controlled by clockwise, anticlockwise, on and off pulses. There are sensors both at the top and the bottom of the gate travel: when the top sensor is active the gate is fully open, when the bottom sensor is active, it is fully shut. The connection to the computer consists of four pulse lines for motor control, two status lines for the gate sensors, and a status line for each class of operator commands. The position of the gate is defined as the percentage of space occupied by the gate: when it is open Position=0, when it is closed Position=1. Finally, the top and the bottom sensors are active when Position becomes less than 0.05 and greater than 0.95, respectively.

The PFs diagram for the sluice gate problem as given in [8] is reported in Figure 1. According to Jackson's clas-



a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling}
c: SO!{Raise, Lower, Stop}

Figure 1. The sluice gate commanded behaviour frame.

sification, the PF in Figure 1 is a commanded behaviour frame, since it identifies a problem where "*there is some part of the physical world whose behaviour is to be controlled in accordance with commands issued by an operator. The problem is to build a machine that will accept the operator's commands and impose the control accordingly*" [8].

The example consists of three domains: the Sluice Controller, which is the machine that will be developed to satisfy the requirements; the Gate & motor, which is the domain to be controlled (it is a causal domain since its proper-

ties include predictable causal relationship among its causal phenomena); the Sluice Operator, which is a biddable domain indicating a user without a positive predictable behaviour (that is, the user can issue commands but cannot be constrained to act in any way).

The next step consists of addressing *frame concerns*, which identify the descriptions that have to be provided with every problem frame. More specifically, addressing frame concerns means making requirements, domain and specification descriptions fit together properly. The combination of these descriptions must result in a 'correctness argument', that is, they must provide evidence that the proposed machine will ensure that the requirements will be satisfied in the problem domain [8]. In the case of the commanded behaviour frame, we have to assure that only sensible and viable commands are executed. Requirements can be expressed as effects on the problem domain caused directly by the user's commands or by other events, such as reaching the completely open (closed) position. According to Jackson, these effects can be expressed in a rather straightforward way by means of state machines.

Also the behaviour of the problem domain can be represented by means of a state machine, showing the states of Gate & motor, and specifying the reactions to external commands, as well as the evolution in time of the domain.

2.3. Representing PFs concepts

In order to support the PFs approach, a modeling language has to satisfy a set of requirements concerning the representation of the relevant concepts according to the definition of the PFs technique [8] and semantics[5].

The fundamental components of problem diagrams and problem frame diagrams are domains and phenomena. Therefore, the language has to be able to represent domains and their characteristics according to the domain type (biddable, causal, lexical), as well as the connections and relations between domains.

Domains are mainly characterized in terms of phenomena, which have to be properly represented. For instance, phenomena of biddable domains are events, while those of causal domains are states and events (generalised as causal phenomena).For every phenomenon it must be clearly specified by which domain it is controlled and by which domains it is visible. As far as shared phenomena are concerned, it is important that sharing is represented without suggesting any communication policy.

Notice that, the same phenomenon may exist in two forms: the real one, which is referenced by the user requirements (e.g., when the heart rate drops below a given threshold, the alarm must be activated), and the sensed one, which is viewed by the machine, and is used in the machine specifications (e.g., when the patient sensor indicates a low heart rate, the machine activates the alarm). The modeling language has to be able to represent both of them, making clear which is the real world phenomenon, and which is the one viewed at the machine interface.

Descriptions can be *indicative*, describing what is given and cannot be changed when developing the solution, or *optative*, describing what has to be achieved by means of the computer-based solution. Both descriptions have to be represented, making clear their differences, possibly by providing different representations.

In PFs, designation of phenomena grounds the problem's vocabulary in the physical world, i.e., it provides names to describe the environment, the system and their artefacts. Therefore a suitable vocabulary must be supported by the modelling language in order to name phenomena.

The modeling language must also support the frame concern which means to support the requirements specification and the representation of the correctness argument.

Finally, since complex problems may not fit into a single PF, the language has to support their composition.

3. A PFs & SysML based methodology

This section introduces the main features and capabilities of SysML, then it presents a modelling approach based on the integration of PFs and SysML.

3.1. SysML: the language

The System Modeling Language (SysML) is a general purpose graphical modelling language for systems engineering applications that has been defined in order to unify the existing notations used by system engineers [15]. The language is a UML2 profile: it inherits some diagrams and other basic elements from UML redefining their semantics and it introduces several new constructs.

The language defines a set of diagram types to describe the structure of a system: the Block Definition Diagram (`bdd`), the Internal Block Diagram (`ibd`) and the Parametric Diagram (`par`).

Block Definition Diagrams extend UML Class Diagrams and describe the static elements that constitute the system architecture. Such elements can be either blocks (basic components) or constraints, which describe relationships and properties among parts of the system. The designer can define standard UML ports and/or flow ports, the latter being ports through which data (or material) can flow continuously. This introduces the possibility to model continuous behaviour, thus overcoming one of the limits of UML 2.

Internal Block Diagrams are an extension of the UML 2 Composite Structure Diagrams used to define instances of the blocks previously defined in a `bdd`. Finally, Parametric Diagrams allow the designer to specify how the constraints defined in a `bdd` have to be allocated to system elements.

The behaviour of the system can be defined by means of Activity Diagrams (`act`), State Machine Diagrams (`stm`), Sequence Diagrams (`seq`), and Use Case Diagrams (`uc`). These are equivalent to the diagrams provided by UML 2,

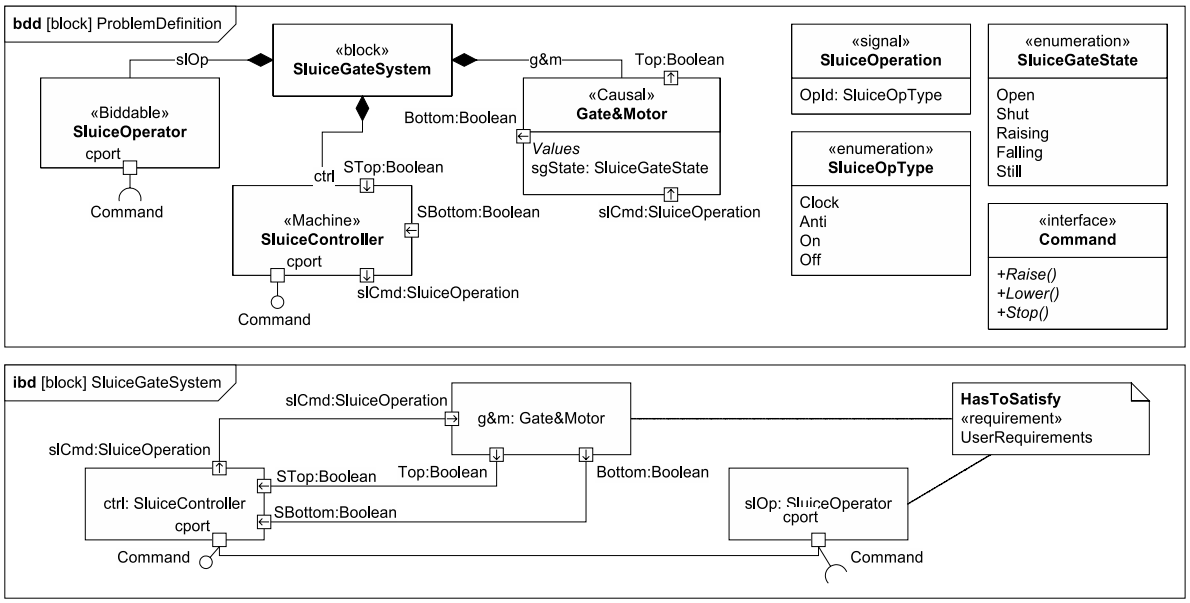


Figure 2. `bdd`, `ibd`**: Representation of the Sluice Gate Control problem.**

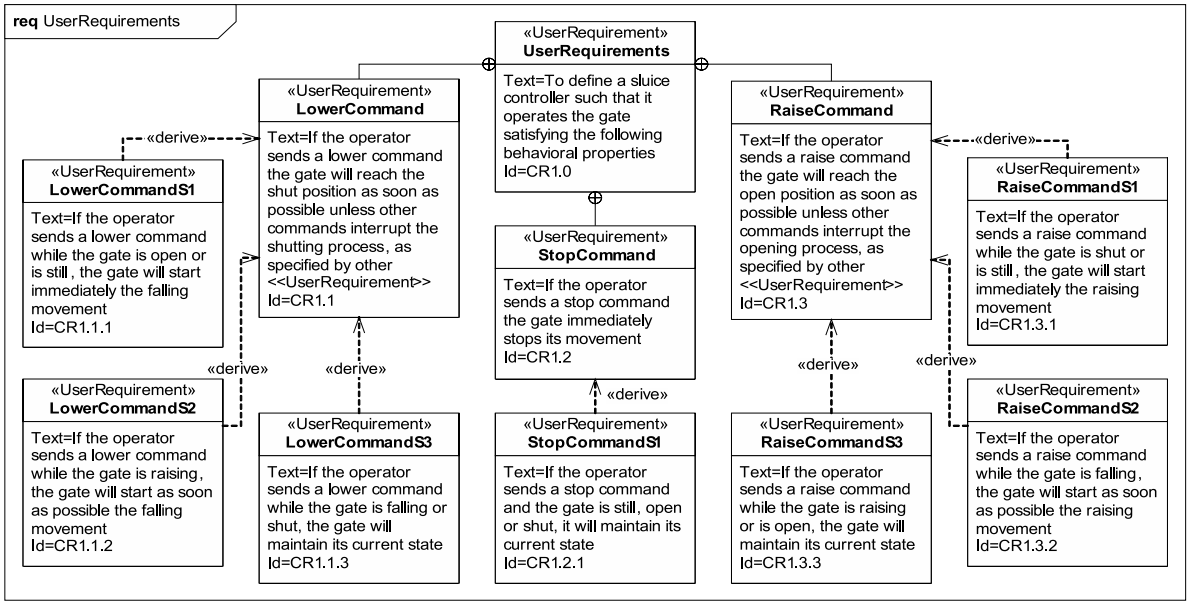


Figure 3. `req`**: User Requirements decomposition.**

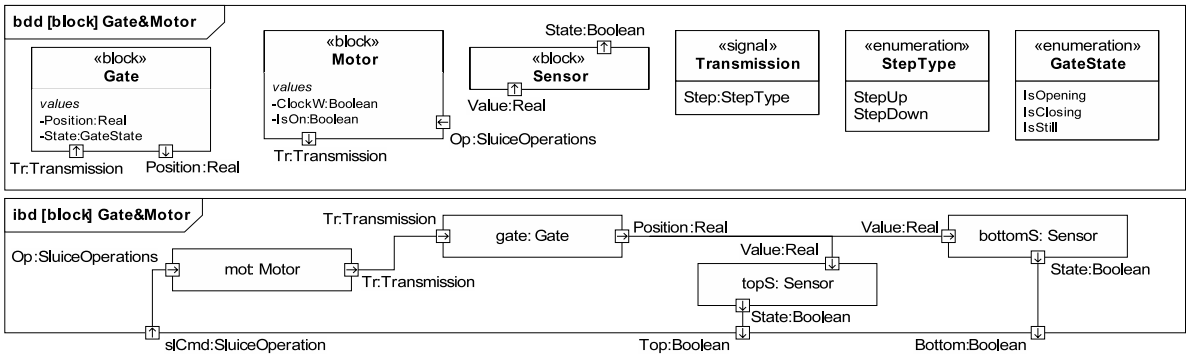


Figure 4. `bdd`, `ibd`**: Decomposition of the** `Gate&Motor` **domain.**

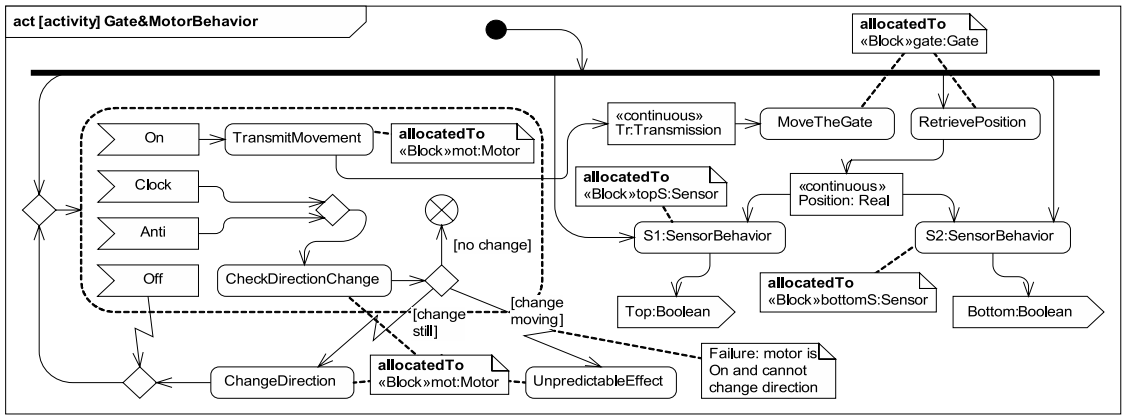


Figure 5. `act`**: The activities associated with the components of the problem domain**

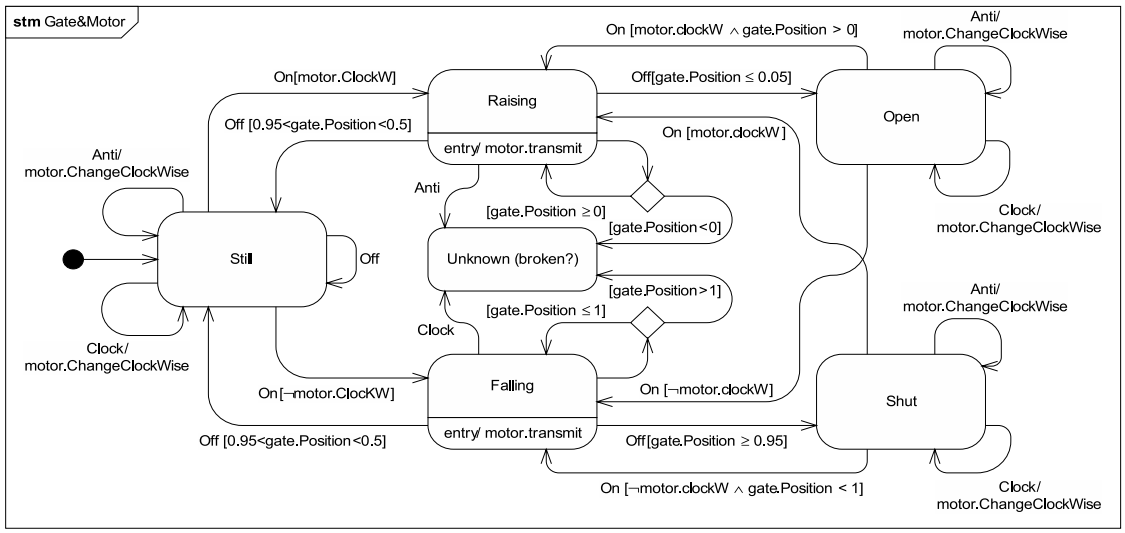


Figure 6. `stm`**: The dynamics of the Gate and Motor**

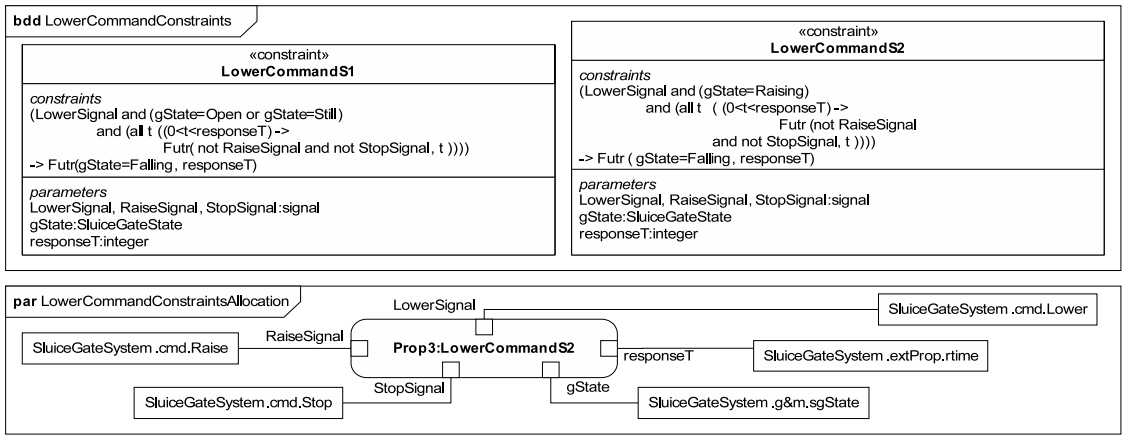


Figure 7. `bdd`, `par`**: Formal specification of the Lower Command requirement.**

except for activity diagrams, which extend the corresponding UML 2 diagram. Activity diagrams support the definition of the system behaviour by specifying the basic activities allocated to the structural elements of the model.

Requirement diagrams (`req`) are a new kind of diagram that is used to define text-based requirements. They allow the analyst to specify functional and nonfunctional properties, as well as constraints that have to be satisfied.

3.2. Representing PFs using SysML

Our approach inherits the conceptual activities from PFs increasing its effectiveness by means of a suitable notation. Moreover, it supports the integration of formal notations for the definition of properties. This allows one to anticipate the verification activities at the requirements engineering time, reducing the risk of expensive design mistakes.

The methodology is structured into the following steps:

- Problem analysis: the problem context, i.e., the *world* in PFs terminology, is decomposed into *domains*, and *shared phenomena* are identified. *Domains* and *Shared phenomena* are represented by means of SysML Blocks and are defined using a `bdd` showing the entities of the problem context.
- Problem definition: the domain blocks are instantiated and connected one to another using an `ibd`.
- Requirements definition: user requirements and properties associated with domains are defined by means of `req` diagrams.
- Domains refinement: domains are refined using SysML diagrams such as `bdd` and `ibd` to support domains decomposition into simpler structures, and `stm`, `act`, `par` and `seq` diagrams to define behaviors.
- Requirements refinement: user requirements are refined by means of `par` and `stm` diagrams.

Notice that such activities are not completely independent one from another and the methodology should not be considered an ordered sequence of steps. Exploiting SysML notation, the involved diagrams provide different views of the same model and thus our methodology exploits a sort of model-centric requirements analysis approach.

The rest of the section discusses in more detail the above steps using the Sluice Gate example.

Problem analysis The domains involved in the problem have to be identified and described, so that requirements can be unanbiguously associated with domains.

Domains are represented by means of SysML Blocks, which can be annotated with stereotypes that specify their nature (e.g., Causal, Machine, Biddable). Shared phenomena are represented by means of SysML Ports and Blocks. Standard Ports describe *event* based communication, while Atomic Ports allow one to access the internal states of Domains. Information is typed by means of Interfaces, Signals or Enumerations. More specifically, *Interfaces* describe events by means of operations. *Signals* define events characterized by complex data structures. *Enumerations* represent the states of a domain. Domains and Shared phenomena are represented by means of a `bdd` diagram describing the entities involved in the problem context. The resulting diagram contains 1) a *system* block representing the *world*, which is composed of domain blocks, and 2) blocks representing the types of the shared information. Domain blocks, in turn, define the ports representing the communication channels through which phenomena can be shared.

The domains of the Sluice Gate Controller problem are specified in the (`bdd`) of Figure 2. The whole system is defined by means of the `SluiceGateSystem` block and its components. The `SluiceOperator` block describes an operator capable of sending specific commands. The `Gate&Motor` block represent the Causal Domain. Both the gate and the motor are given elements. The `SluiceController` represents the Machine Domain: the properties of its interaction with the environment must be defined in order to satisfy the user requirements.

All the aforementioned domains specify interaction points through which phenomena can be shared. As an example, the `SluiceOperator` requires the `Command` interface to allow the operator to issue commands. The interface is actually provided by the `Controller` block. Similarly, the `Gate&Motor` block defines two atomic output ports that propagate the events generated by the sensors according to the position of the gate. Moreover, the `Controller` defines two dedicated input ports to receive notification by external sensors.

Notice that shared phenomena are specified by defining the types associated with ports and interfaces: for instance, the `SluiceOperation` signal specifies the nature of the commands sent from the Machine to the `Gate&Motor`.

Problem definition PFs Problem Diagrams are represented by means of `ibd` diagrams that define the internal organization of the *system* block. Such diagrams define how the domains are instantiated and connected one to another.

The `ibd` of Figure 2 describes the context of the SluiceGate problem showing how the involved domains share information. For instance, the `SluiceOperator` and the `SluiceController` share phenomena that are directly generated and controlled by the `SluiceOperator` and received by the `SluiceController`.

Requirements definition Requirements are categorizable in two distict classes: 1) *User requirements*, which specify the user expectations against the solution of the problem,

and 2) *System requirements*, which specify properties associated with domains. *User requirements* are defined using a `req` diagram; they are specified by means of informal textual descriptions structured in a sort of hypertext; decomposition and derivation relationships allow one to organize and refine requirements. Figure 3 reports the `req` diagram of the Sluice Gate problem.The behaviour of the causal domain `Gate&Motor` is constrained according to the phenomena generated by the biddable `SluiceOperator` domain: for instance, User requirements specify when commands have to be considered useless, and the expected results in such cases.

Once defined, *User requirements* have to be allocated to problem domains by annotating the correspondent blocks of the `ibd`. For instance, as shown in Figure 2, `Gate&Motor` and `SluiceOperator` are the involved domains.

System requirements express properties concerning domains. As in the previous case, they can be defined and structured using `req` diagrams. However, this is not mandatory, since `req` diagrams can be replaced by other diagrams that could better support the environment description as explained in what follows.

Refining domains descriptions Domains represent the constituent parts of the *world*. They are introduced in `bdd` and `ibd` diagrams that describe the structural aspects of the problem context at an high level of abstraction. Moreover, domains properties are possibly informally described in the `req` diagram that defines the *System requirements*. Domains descriptions are now refined into more precise and detailed specifications.

System requirements are characterized by static and dynamic aspects, and thus depending on their nature they have to be refined exploiting different diagrams. Structural properties are defined by means of `bdd` and `ibd` diagrams, while behaviors are described by means of `act`, `stm` and `par` diagrams. SysML, having been conceived to model potentially complex systems, provides (de)composition mechanisms for all diagrams and elements, thus making it possible to structure the system descriptions at different levels of aggregation and abstraction.

For instance, Figure 4 shows the internal structure of the `Gate&Motor`. Domains decomposition follows the same criteria applied for the problem definition (i.e., sub-domains are defined by means of Blocks in a `bdd`, and instantiated and used in a `ibd`, where the connections are shown).

`stm` are used to specify in an operational style the behavior associated with a single Block. For instance, the `stm` of Figure 6 specifies the reactions to external commands and the internal evolution of the `Gate&Motor` domain.

`act` are used whenever it is necessary to specify a complex behaviour involving several domains. Also these diagrams feature an operational style and support the composition of simpler activities. In our example, the coordinated behaviour of the Gate and Motor pair is specified by means of an activity diagram, as shown in Figure 5.

Refining requirements The `req` diagrams defining the *User requirements* can be used by the analyst to define and organize requirements in an informal top-down apprach. However in order to apply automatic verification techniques such properties need to be expressed in a formal way.

Our approach does not impose the usage of a specific formal language for refining purposes. The modeler is free to adopt the language that he/she considers most suitable to the problem. The methodology simply defines *how* formal properties have to be specified and integrated in the model.

`bdd` have to formally define properties with `constraint` blocks. Constraints are expressed by composition of parameters. `par` diagrams allocate constraints to one or more (sub)domains. Notice that, the same approach can be used also for refining domain descriptions. Figure 7 shows a formalization of a *User requirement* (the `Lower` command) using TRIO, a first order temporal logic for the specification of real-time systems [4].

`stm` represent an effective alternative to the specification of constraints. They are particularly suitable to the definition of properties concerning the evolution of blocks. Their usage is widely supported by different analysis techniques that support formal verifications [2].

4. Related work

The proposed approach for integrating SysML and PFs can be seen from two perspectives: 1) providing SysML with a rigorous conceptual framework and a set of methodological guidelines, and 2) providing PFs with a suitable notation.

In the former case, there are no proposals concerning how to use the language. In fact, the literature reports just a few experiences in using SysML. For instance, some whitepapers are available from Artisan, describing systems like a house heating system [14] or a waste treatment plant [6]. However, these papers describe the nature and the role of SysML diagrams, rather than suggesting methodological guidelines.

A SysML modelling approach for real-time systems is proposed in [3]. An experimental application of SysML in the real-time domain showed that SysML supports well the definition/usage dichotomy (e.g., concerning the `bdd`/`ibd` and `bdd`/`par` pairs), the hierarchical decomposition of behavioral and static elements, and the usage of cross-cutting constructs (namely requirements and allocation mechanisms). In addition, the paper discusses the limits of the language, mostly related to the lack of formal semantics and mechanisms to extend and adapt the language. However, these limits can be partially overcome by using formal languages for defining constraints and properties.

The original proposal of the PFs approach [8] already addressed the methodological issues of requirements engineering. On this basis, several researchers built extensions and refinements of the method. Some concentrated on the adaptation for specific purposes, like web service requirements modelling [10], or e-business modelling [1]. Others studied the transition from requirements to machine specifications [13, 18]. Jackson and co-authors also elaborated on the PFs approach in several directions, such as defining the semantics of PFs [5], composing PFs [11], and highlighting the role of PFs in software engineering [9]. However, none of the mentioned works concerned the linguistic issues, nor the integration of PFs in any popular model-based development practice.

5. Conclusions

The goal of the work presented in this paper was to integrate the Problem Frames requirement analysis approach with the SysML notation. The experimental application of this integrated approach showed that PFs can be effectively described by means of SysML diagrams, and that PFs concepts can be effectively supported by SysML.

A relevant benefit of using SysML is represented by the description of the problem domains, that are mostly composed of non-software entities; note that this would hold even if we used a different requirement analysis methodology. SysML supports the definition and management of parametric constraints in a direct way, allowing the analyst to express properties and requirements by means of formal languages. Another important feature of SysML is the ability to deal with continuous behaviour, which often characterizes physical problem domains. For instance, in the example presented, the communication between the motor and the gate models a physical transmission, which maintains a continuous connection between the involved domains. SysML provides constructs to model phenomena –such as entities, signals, events, flows, etc.– in a concise and expressive way. This applies to both phenomena that are private to a domain and to shared phenomena.

Under all these respects, SysML proved to be more expressive and flexible than UML [12]. Among the limitations of UML that are overcome by SysML are the design-oriented component diagrams, which tend to force the specification of policies defining interfaces, the lack of support for modelling continuous behaviour, the difficulty of including formal specifications in the models.
In the experimental modelling of the sluice gate controller we followed the methodology suggested by Jackson in [8] and we found that the specification practice was smoothly supported by SysML.

The work we carried out suggests a set of objectives for future activities, such as the definition of a SysML profile inluding ready-to-use PFs concepts, the need to study and experiment with the extensions of the PFs approach to System Engineering, and the execution of a real case study involving the development of systems composed of both software and hardware parts.

References

[1] S. Bleistein, K. Cox, and J. Verner. Problem Frames approach for e-business systems. In *Int. Work. on Advances and Applications of Problem Frames*, 2004.

[2] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[3] P. Colombo, V. D. Bianco, L. Lavazza, and A. Coen-Porisini. An experience in modeling real-time systems with SysML. In *Proc. of Int. Work. on Modeling and Analysis of Real-Time and Embedded Systems*, 2006.

[4] C. Ghezzi, D. Mandrioli, and A. Morzenti. Trio: A logic language for executable specifications of real-time systems. *Int. Journal of Systems and Software*, 12(2), 1990.

[5] J. G. Hall, L. Rapanotti, and M. Jackson. Problem frame semantics for software development. *Software and Systems Modeling*, 4(2), 2005.

[6] M. Hause and F. Thom. *Building Embedded Systems with UML 2.0/SysML*, 2005. ARTiSAN Software, Whitepaper.

[7] M. Jackson. *Software Requirements and Specifications*. ACM Press Books, 1995.

[8] M. Jackson. *Problem Frames - analysing and structuring software development problems*. Addison-Wesley ACM Press, 2001.

[9] M. Jackson. Problem Frames and Software Engineering. In *Proc. of Int. Work. on Advances and Applications of Problem Frames*, May 2004.

[10] A. Jha. Problem frames approach to web services requirements. In *Proc. of Int. Work. on Advances and Applications of Problem Frames*, 2006.

[11] R. Laney, L. Barroca, M. Jackson, and B. Nuseibeh. Composing requirements using Problem Frames. In *Proc. of Int. Conf. on Requirements Engineering*. IEEE CS Press, 2004.

[12] L. Lavazza and V. D. Bianco. Combining Problem Frames and UML in the description of software requirements. In *Proc. of Int. Conf. on Fundamental Approaches to Software Engineering (FASE06)*, 2006.

[13] Z. Li, J. G. Hall, and L. Rapanotti. From requirements to specifications: a formal approach. In *Proc. of Int. Work. on Advances and Applications of Problem Frames*, 2006.

[14] A. Moore. *SysML Hits the Home Straight*, 2006. ARTiSAN Software Tools, whitepaper.

[15] OMG. *OMG System Modeling Language (OMG SysML) Specification*, May 2005. Final Adopted Specification, ptc/06-05-04.

[16] OMG. *Unified Modeling Language: Superstructure*, 2005. Ver. 2.0, formal/05-07-04.

[17] OMG. *Unified Modeling Language: Infrastructure*, 2006. Ver. 2.0, formal/05-07-05.

[18] R. Seater and D. Jackson. Problem Frame transformations: Deriving specifications from requirements. In *Proc. of Int. Work. on Advances and Applications of Problem Frames*, 2006.