HAL
open science

Bridging the Gap Between Requirements Document and Formal Specifications using Development Patterns

Imen Sayar, Jeanine Souquières

To cite this version:

Imen Sayar, Jeanine Souquières. Bridging the Gap Between Requirements Document and Formal Specifications using Development Patterns. IEEE 27th International Requirements Engineering Conference Workshops (REW), Sep 2019, Jeju Island, South Korea. 10.1109/REW.2019.00026 . hal-02962897

HAL Id: hal-02962897

https://hal.univ-lorraine.fr/hal-02962897v1

Submitted on 9 Oct 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Bridging the Gap Between Requirements Document and Formal Specifications using Development Patterns

Imen Sayar
Université de Lorraine, CNRS, LORIA
F-54000 Nancy, France
imen.sayar@loria.fr

Jeanine Souquières
Université de Lorraine, CNRS, LORIA
F-54000 Nancy, France
jeanine.souquieres@loria.fr

***Abstract*—Guaranteeing the correctness of critical and complex software and systems is a challenge that needs to be tackled right from the requirements engineering phase. This paper introduces two development patterns linked to the shape of requirements. The first one allows to automatically formalize a constraint and introduce it in an existing system. The second one is interested on requirements describing a sequence of operations. The verification activity is partly automated and the validation becomes easier to manage. The approach using these development patterns allow us an incremental development of formal specifications and their associated requirements, linked by a glossary. The case study of a hemodialysis system is used as a running example throughout this paper.**

I. Introduction

A requirements document serves as a bridge between the clients and suppliers of software and systems development industry. The development of formal specifications is a manual activity based on the cognitive skills of the person in charge of developing them out of the informally described user requirements. Although there has been considerable focus on making requirements more understandable by reducing the ambiguity, there is hardly any support for the development of formal specifications. Approaches that propose the use of controlled natural language for the description of requirements improve the clarity of the requirements, but do not contribute directly to the development of formal specifications.

Patterns are used in software programming to document solutions, facilitating their application to new problems [7]; they are templates for how to solve a problem that can be used in many different situations. Patterns are also used in software specifications, describing recurrent specification structures [9].

We propose development patterns to formalize requirements describing two concepts: constraints and sequences. A pattern uses a system of already developed formal specifications, their corresponding requirements and the trace links between them. They automate the development of parts of the formal specification. This work is an evolution of our previous work published in [15].

The remainder of the paper is organized as follows. First we present our approach in Section II with utilisation of different existing tools. The first pattern presented in Section III concerns the formalization of constraints described in the requirements and Section IV concerns the definition of a sequential pattern. We applied our patterns on several case studies as the landing gears sytem [3], the hemodialysis [13] and the Hybrid ERTMS/ETCS Level 3 standard[1] [4]. In this paper, we use the hemodialysis machine case study. Section V discusses our contribution. Section VI compares our work with the existent and finally, Section VII concludes and sketches future work directions.

II. The approach

A. Description

Our work is situated in the context of bridging the gap between two different levels of formalisms *i.e.* the requirements document of the client and the formal specification expressed by the computer specialist. The first document describes informal or semi-formal artifacts used to describe user's point-of-view of the system under development and the second one represents artifacts describing the developer's point-of-view of the same system in a formal manner. A system is defined by:

- *Reqs.* They represent a list of rewritten user requirements using Abrial's approach [2]. Our methodology is not dependent on a specific tool, however we use the ProR tool [11] for documenting the requirements.
- *Spec.* It denotes the specification of the future system, described by a formal method approach like the Event-B method based on the refinement technique. An Event-B specification is composed of two constructs:
  - the *context* describes the static part of the model, using sets, constants, theorems and axioms,
  - the *machine* describes the behavior of the model. It contains the system variables, invariants and events. An event is the dynamic element of the machine. It is composed of guards and actions.
- *Glossary.* It describes the traceability links between the two previous documents associating the formal terms

[1] https://www.southampton.ac.uk/abz2018/information/case-study.page

coming from the *Spec* to their corresponding informal descriptions in the *Reqs*.

The development patterns arise from the informal requirements. They are used to perform activities of the specification development by simplifying the verification and validation activities. A development pattern concerns the three components of the $< Reqs, Glossary, Spec >$ system. We present two patterns, *Dev-if* and *Dev-seq*. The first one serves on automatically introducing constraints in an existing system. The second one aims on automatically defining order between unordered operations of a system. The constraints and the sequences are described in the informal client requirements document.

B. Tools

1) Managing requirements: We use *ProR*, plug-in of Rodin[2] to edit requirements and link them with formal specifications. Requirements can be organized in a hierarchical structure following:

- a *brother-brother* relation in which a requirement is in the same level as a *"brother"* requirement or
- a *parent-child* relation where a *"child"* requirement details a *"parent"* one, see Figure 9.

Three requirements of the hemodialysis machine case study [13] are shown in Figure 1.

ID	*Description*
R-5	During initiation, if the software detects that the pressure at the VP transducer exceeds the upper pressure limit, then the software shall stop the BP and execute an alarm signal.
R-6	During initiation, if the software detects that the pressure at the VP transducer falls below the lower pressure limit, then the software shall stop the BP and execute an alarm signal.
R-8	During initiation, if the software detects that the pressure at the AP transducer falls below the lower pressure limit, then the software shall stop the BP and execute an alarm signal.

Fig. 1. Some requirements from the hemodialysis case study

2) Verification: It concerns the specification and answers the question "*are we constructing the system correctly ?*". The feedbacks of this activity give indications about shortcomings in the requirements document like contradictions or oversights [1]. The semantics of specifications are given by proof obligations (POs) ensuring that machines meet essential system properties, such as safety or invariant-preservations. The POs are generated by proof obligation generators and discharged using the automatic and interactive provers of the Rodin platform. The correctness of the specification of the patterns has been proved once and for all. The verification task is automatically done for the Event-B specifications in the same manner as proposed in [9].

3) Validation: It checks if the developed specification is coherent relatively to the client requirements and focuses on responding to the question "*are we developing the correct system ?*". ProB [12] is used for the animation and model-checking of the specification. In [16], we have proposed an approach for validating the formal specification with respect to the user requirements, using the glossary, representing the links between these two documents. The validation starts from the requirements analysis phase during which we extract validation elements from the client requirements. A validation element refers to formal or informal terms according to which the *Spec* will be validated. It concerns:

- the *data* that should be present in the system,
- the expected *functionalities* or services provided by the system,
- the *conditions* or *obligations* under which the system works. They are compared to preconditions and post-conditions of Hoare [10] and
- the *behavior* defined using existing functionalities and described as a sequence of operations or services.

[2]A platform based on Eclipse and available at http://www.event-b.org/

III. Conditional pattern

It formalizes informal constraints - described in client requirements - into formal elements in an existing system.

A. Problem

Given a system $< Reqs, Glossary, Spec >$ and a requirement *R-new*, the problem is how to automatically introduce a constraint in this system. The resulting system should be correct. Then, the effort of developing is gained and the risk of oversights is reduced. We define a requirement describing a constraint as follows:

$$\textit{R-new:} \quad \textit{env_vars}\ \textbf{if}\ condition\ \textbf{then}\ action$$

where:

- *env_vars* is a set of values of variables describing the state of the environment of the user requirement,
- *condition* is a boolean expression describing the constraints on variables of the system and
- *action* designates the instructions that modify the state of system variables.

B. Solution and formalization

The development pattern *Dev-if* operates on the requirement R-new that describes a constraint on the system's functioning, see Figure 2.

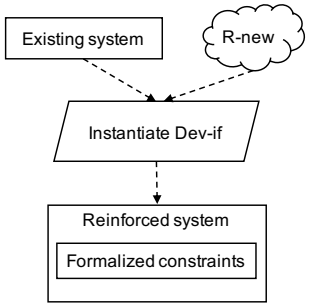


Fig. 2. Dev-if instantiation

1) Reqs: The requirements document evolves by introducing *R-new′*, a formalized form of the original *R-new*. We mention that this latter is kept in the requirements document for traceability reasons and is preceded by an *. *R-new′* contains formal terms - put between square brackets *[ ]* - coming from the specification and replacing the informal terms.

ID	*Description*
* R-new	env_vars **if** condition **then** action
R-new'	[env_vars_val] **if** [condition] **then** [action_vars]

2) Spec: It is described by two concepts:

- *The invariants. Dev-if* formalizes system properties via predicates:
  - *R-new′-prop1* is an invariant describing the constraint described in *R-new*. This invariant means that if the system's action variables change their values, then one can be sure that this system ensures the imposed condition and respects the environment variables values, see Figure 3.
  - *Dev-if* ensures the preservation of the abstract properties of the formal specification by its concrete version with introducing the gluing invariant *glue-R-new′*. "..." expresses the parts that should be completed by the developer.

  NB. The operator "∧" expresses the conjunction between abstract and concrete states of variables and the exclusive disjunction operator "⊕" describes the exclusion between these states. The expertise of the developer is needed to accomplish this task.

$$R-new'-prop1 : (action_vars = values)^{+} \Rightarrow condition \wedge env_vars = env_vars_val$$
$$glue-R-new' : (action_vars = values)^{+} \wedge / \oplus \ldots \Rightarrow condition \wedge env_vars = env_vars_val \wedge / \oplus \ldots$$

Fig. 3. Invariants of conditional pattern

- *The events.* The formalization of *R-new* is defined by the event *treatment_R-new'* of Figure 4 expressing:
  - the constraint on the environment variables state using the guard *grd1*,
  - the condition using the guard *grd2* and
  - the action via the assignment *R-new'-act*. This assignment is described by a deterministic action that can be repeated, hence the use of the symbol "+".

Event $treatment_R-new'$
  when
    $grd1 : env_vars = env_vars_val$
    $grd2 : condition$
  then
    $R-new'-act : (action_vars := values)^{+}$
  end

Fig. 4. Event of the conditional pattern

3) Glossary: New formal terms attached with their informal descriptions are automatically added to the glossary [8].

C. Activities

1) Verification: The proposed pattern is described as an element of an Event-B specification. Six POs related to this pattern are automatically generated using the tools associated to the Rodin platform. When applying it to an existing system, these POs will be automatically instantiated and discharged since the prooving activity is already done for the pattern.

2) Validation: The *Dev-if* pattern offers elements related to the validation activity. These elements are generic, that means they will be automatically instantiated when applying the pattern. They concern:

- *data* like the *action_vars* representing the variables involved in the action of the condition and the *env_vars* describing the variables of the system environment,
- *functionalities* such as the event *treatment_R-new′* which will be added to the *Spec* and
- *obligations* as the invariant *R-new′-prop1* (see Figure 3) constricting the system's functioning. This obligation is generated and checked automatically when applying this pattern.

D. Case study

The hemodialysis case study is presented by a technical part and a safety requirements part. This latter is composed of general requirements and 36 software requirements. These last are almost describing constraints in the system components with a repetitive way. Our starting point is described by the following system[3]:

- *Reqs.* They are representing the informal requirements document containing R-5', a rewritten form of R-5. We use the recommendations of Abrial [2] and the ProR tool [11] to realize this requirement. R-5' contains formal terms coming from the *Spec*. The *Reqs* document is described as follows:

ID	*Description*
...	...
* R-5	During initiation, if the software detects that the pressure at the VP transducer exceeds the upper pressure limit, then the software shall stop the BP and execute an alarm signal.
R-5'	[initiat] if [vp] exceeds [upper_press_limit] then stop [BP] and execute [ALM_excess_vp]

- *Spec.* It is described by the machine R-5′_Mch which refines the machine Common_Mch and sees a context R-5′_Ctx. An overview of this specification is provided in the Figure 5.
- *Glossary.* It contains the available pairs:

Formal term	*Informal description*
initiat	During initiation
vp	the pressure at the VP transducer
ALM_excess_vp	an alarm signal
...	...

[3]The complete development is available in our team website http://dedale.loria.fr/

Context $R-5'_Ctx$
Sets
$ALARMS$
Constants
$NULL, ALM_excess_vp$
Axioms ...

Machine $R-5'_Mch$
Refines $Common_Mch$
Sees $R-5'_Ctx$
Variables
$vp, alarm_vp, BP$
Events
$INITIALISATION$
$manage_excess_vp$
$\ldots$

Fig. 5. Overview of the specification of the requirement R-5

1) Introducing a condition: Let's start the development of the requirement R-6 relatively to the existing system including the development of R-5 (see Figure 1). R-new corresponds to R-6. Table I shows the values of the parameters in the requirement in question.

TABLE I
PARAMETERS VALUES IN R-6

Parameter	***Value***
env_vars	During initiation
$condition$	the pressure at the VP transducer falls below the lower pressure limit
$action$	- stop the BP - execute an alarm signal

This requirement uses the variable *vp*, *the pressure at the VP transducer*. The *Dev-if* pattern takes as inputs: the existing development of R-5 and R-6 requirement. It updates the system by new details coming from R-6:

- *Reqs.* The rewritten requirement *R-6'* is added to the existing requirements document. It is described in Figure 6 as a *brother* of both R-5' and R-6. The two requirements have the same variable *vp*.

ID	*Description*
...	...
* R-5	During initiation, if the software detects that the pressure at the VP transducer exceeds the upper pressure limit, then the software shall stop the BP and execute an alarm signal.
R-5′	[initiat] if [vp] exceeds [upper_press_limit] then stop [BP] and execute [ALM_excess_vp]
* R-6	During initiation, if the software detects that the pressure at the VP transducer falls below the lower pressure limit, then the software shall stop the BP and execute an alarm signal.
R-6′	[initiat] if [vp] falls below [lower_press_limit] then stop [BP] and execute [ALM_deficit_vp]

Fig. 6. Updated requirements document

- *Spec.* A new event *treatment_R-6′* is automatically generated and added to the *R-5′_Mch*:

Event $treatment_R-6'$
when
$grd1 : phase = initiat$
$grd2 : vp < lower_press_limit$
then
$R-6'-act1 : BP := stopped$
$R-6'-act2 : alarm_vp := ALM_deficit_vp$
end

Note that the new constant *ALM_deficit_vp* is added to the context *R-5′_Ctx*.

- *Glossary.* It is updated by adding a new pair *(ALM_deficit_vp, an alarm signal)*.

The verification and validation activities are fulfilled as follows:

- the POs associated to this specification are automatically discharged and
- the validation is performed using the tool ProB. We successfully animate the updated machine *R-5'_Mch* using the following scenario:

$$INITIALISATION \rightarrow start_BP \rightarrow change_vp \rightarrow treatment_R-6'$$

in which the events *start_BP* and *change_vp* refer to respectively the operations of starting the blood pumping and changing the value of the pressure at the VP transducer of the hemodialysis machine.

2) Introducing the glue: We refine the development issued from Section III-D1 by taking into account the requirement R-8 (see Figure 1). The *Dev-if* pattern ensures the preservation of the abstract *Spec* previously described and automatically generates new elements. It updates the requirements document by a rewritten form of R-8 and updates the glossary.
Dev-if generates "..." which concerns the concept glue between R-8 and R-6. This concept expresses the state of the system when both *the pressure at the AP transducer* and *the pressure at the VP transducer* fall below *the lower pressure limit*. The developer completes the "..." by:

- *Reqs.* A requirement *glue-R-8′* is added. Hereby, a completed version of this requirement is a brother of both R-5′and R-6′.

ID	*Description*
glue-R-8′	[initiat] if [ap] falls below [lower_press_limit] and [vp] falls below [lower_press_limit] then stop [BP], execute [ALM_deficit_ap] and execute [ALM_deficit_vp]

- *Spec.* A gluing invariant is introduced.

$$glue-R-8' : \quad alarm_ap = ALM_deficit_ap \wedge alarm_vp = ALM_deficit_vp \Rightarrow ap < lower_press_limit \wedge phase = initiat \wedge vp < lower_press_limit$$

- *Glossary.* There is no new pairs added.

The verification and validation of this system are automatically performed using the Rodin tools.

IV. SEQUENTIAL PATTERN

This pattern helps the developer to automatically introduce the order between existing operations of a given system. It concerns the formal Event-B specification and the requirements and the glossary.

A. Problem

In the documents of the three case studies previously mentioned (see Section I), the requirements describe a sequence of operations using this form:

$$\textit{R-new:} \quad \textit{(env_vars)*}\ \textbf{sequence}\ (operation)^{+}$$

in which:

- *env_vars* is a set of variables describing the environment of the requirement. The asterisk following this element means that these variables may not exist and
- *operation* represents a series of ordered actions. It contains at least one action.

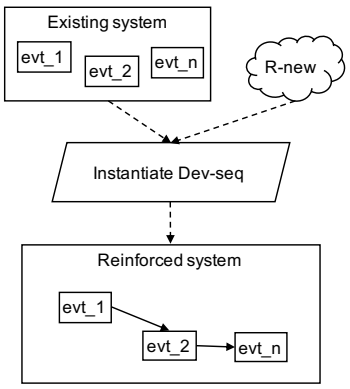
Given an existing system containing unordered operations and a requirement R-new describing a sequence, the problem is how to automatically introduce order between this system's operations.

B. Solution and formalization

We formalize the pattern *Dev-seq* in Event-B using the Rodin platform and in TLA^{+} using TLA Toolbox[4], an integrated development environment. Using Event-B language, every $operation$ of *R-new* is translated into an event. The order is expressed by the guards and the assignments in the events: the assignment of an event is the guard of the next one. Comparing with the Hoare triplet [10] ($precondition\ \{instructions\}\ postcondition$), each action in a sequence has:

- a precondition representing the result of the previous action and
- a postcondition resulting from its execution.

Dev-seq introduces the order between the operations of an existing system. Its use is described in the following figure. It takes as inputs the existing system composed of several unordered operations or events $evt_i \mid i \in \{1..n\}$ and a requirement *R-new* describing the sequence. The generated output is a system strengthened by taking into account the order between the previous events/operations.



[4]https://lamport.azurewebsites.net/tla/toolbox.html

C. Activities

Verification and validation activities are automatically performed using *Dev-seq*. This pattern generates scenarios describing a chain of ordered operations/events. For instance,

$$INITIALISATION \rightarrow \ldots \rightarrow evt_1 \rightarrow \ldots \rightarrow evt_n$$

is one of these generic scenarios.

D. Case study

We treat the following requirement R-th of Figure 7 taken from the section "*Connecting the patient and starting therapy*" of [13]:

ID	*Description*
R-th	- The patient is connected arterially. - The BP is started by pressing the START/STOP button on the UI. - The blood flow is set. - The blood tubing system is filled with blood. The BP stops automatically when blood is detected on the VRD in the SAD. - The patient is connected venously. - The blood pump is started and the prescribed blood flow is set. - The machine is taken out of bypass mode. The HD machine switches to main flow and bicarbonate running. The signal lamps on the UI switch to green.

Fig. 7. Informal requirement describing a therapy

This requirement describes implicitly a sequence. In order to explicit this sequential form, we re-write it as follows:

$$\textit{R-th:} \quad \textbf{\textit{sequence}}\ \textit{therapy}$$

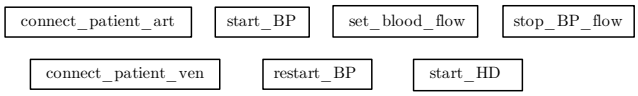
in which $therapy$ is the series of actions presented by each item of Figure 7. The env_vars parameter is empty.

1) Existing system:

- *Reqs.* They represent the informal requirements containing *R-th'*, a rewritten form of *R-th*, in which formal terms are introduced between brackets "[ ]". This requirement is partially described as follows:

ID	*Description*
R-th′	**sequence** - [patient] is [connected_art]. - [BP] is [started] by pressing the START/STOP button on the UI. - [blood_flow] is [set]. - ...

- *Spec.* It is described by the following seven events. Each event formalizes one sentence of *R-th*. There is no order between them.



- *Glossary.* Some pairs of (formal term, informal description) are shown in the following:

Formal term	*Informal description*
...	...
patient	The patient
connected_art	connected arterially
BP	The BP
blood_flow	The blood flow
set	set

2) Taking into account the order: Dev-seq introduces an order between the events of the previously described *Spec*. Their updated version is shown in Figure 8 where:

- the result of *act1* of event *connect_patient_art* is a guard *grd1* of the event *start_BP* and
- the result of *act1* of event *start_BP* is a guard *grd1* in the event *set_blood_flow*.

Event *connect_patient_art*
then
act1: patient := connected_art
end

Event start_BP
when
grd1: patient = connected_art
then
$act1 : BP := started$
end

Event set_blood_flow
when
$grd1 : BP = started$
then
$act1 : blood_flow := set$
end

Fig. 8. Introducing an order between events

Thanks to the concepts of *guards* and *actions* in the events, these ones are executed as a chain:

connect_patient_art → start_BP → set_blood_flow

The requirements document is updated in the Figure 9 by:

- decomposing *R-th'* into sub-requirements and
- introducing an order between sub-requirements using *numbers*.

ID	*Description*
R-th′	**sequence**
R-th′-1	**1-** [patient] is [connected_art].
R-th′-2	**2-** [BP] is [started] by pressing the START/STOP button on the UI.
R-th′-3	**3-** [blood_flow] is [set].
...	...

Fig. 9. Updated requirements document

The verification and validation activities are realized automatically. For example, the sub-scenario

$$connect_patient_art \rightarrow start_BP \rightarrow set_blood_flow$$

associated to R-th' and some of its children is instantiated by *Dev-seq*. Using the ProB tool, the resulting machine animates successfully this scenario.

V. Recapitulations and lessons learned

These two patterns automate parts of the development:

- *Dev-if* allows taking into account constraints of a system. It updates an existing system by staying in the same level or refining it.
- *Dev-seq* introduces an order between the operations or events of an existing system.

Using these patterns for the landing gears system and the hemodialysis case studies, we obtain results shown in Table II.

- **The landing gears system:** four models of the formal specification are developed using the *Dev-seq* pattern. In total, 310 POs are discharged for these specifications in which 256 are automatic, thus the 82 percent of the total POs.
- **The hemodialysis machine:** nine specifications are developed using *Dev-if* for which 301 POs by 341 are discharged automatically.

TABLE II
Results of application of *Dev-if* and *Dev-seq*

	Landing gears	Hemodialysis
Number of Specs	4	9
Automatic POs	256	301
Total of POs	310	341
Percentage	82	88

The application of these two patterns allows us to be aware of the importance of the existence of tools for managing requirements, verifying and validating the formal specifications. Without these tools, the task of using our patterns is hard to accomplish.

This work is a beginning of a future project for constructing a library for development patterns. While working on these patterns, several questions are arising:

- In the *Reqs*. How to introduce the rewritten forms of the original requirement ? Is it a brother, a father or a son of the other requirements ?
- In the *Spec*. How a new requirement will be integrated ? What kind of efforts and of skills should have the developer when applying these patterns ? How can the pattern decide whether it adds or not the glue ?
- In the *Glossary*. How can this document help to ensure the "good" application of the development patterns ? Does it allow to detect errors generated by the patterns in the *Reqs* and the *Spec* ?

The *Glossary* is important in our approach. It allows linking permanently the formal elements with their informal description in the requirements. The trace descried by this document helps to facilitate the access to elements of the *Reqs* while validating the *Spec*. The *Glossary* is described in our patterns and is instantiated when applying them.

VI. Related work

A pattern enables the description of an identified subproblem and its solution by reusing knowledge acquired through experience. Our idea is to predefine development solutions and incorporate them in the requirements document, using Event-B and the Rodin platform. They allow the reuse

of the specification models and their correctness in terms of proofs. Hoang et al. [9] define patterns for Event-B specification in order to reuse an existing formal model and to reduce the proving effort. We have exploited the patterns emerging from the writing styles of the requirement descriptions.

KAOS [17] proposes a goal-oriented approach for requirements modeling and refinement. Its enables the identification of the system goals and their gradual refinement until obtaining constraints using formal refinement patterns. [14] consider the interactions between the artifacts of the requirements. [6] define a preliminary work on a language dedicated to combine requirements with the formal specifications. They use several approaches like KAOS and OCL to describe the requirements and to their formalization. In our approach, we make the evolution of the three components (Reqs, Spec and the Glossary) at the same time. We use tools such as ProR, ProB, provers of the Rodin Platform at any moment of the development.

The authors of [5] define three refinement patterns for algebraic state-transition diagrams (ASTDs). These patterns are complementary to the specification patterns [9] by gradually introducing details in the specification using refinement. The authors compare their patterns with CSP and Event-B refinement. Our patterns are used for requirements and specifications and take into account the refinement by automatically generating the glue between abstract and concrete models.

VII. Conclusion and future work

This paper presents an approach based on well-proven generic scheme, the development patterns used to write down predefined forms for the requirements. These patterns concern informal requirements described in a repetitive form, a conditional one and sequential one. They take into account the refinement technique and are proved complete and correct. We demonstrate our approach by applying it on the hemodialysis machine case study and on the landing gears system. Tools incorporated in the Rodin platform are used in all the development steps: ProR for managing requirements and their links with the formal specification, automatic and interactive provers for checking the correctness of the specification, ProB for animating and model-checking the formal models and graphical tools like Event-B state machine[5] and Project Diagram[6] for showing machines and contexts relations.

Dev-if is formalized in the Event-B language. *Dev-seq* is formalized in both Event-B and TLA^+ in order to prove that our patterns are independent of the specification language. One limitation of our patterns is that their use is still not automated; it is done manually. As a future work of this approach, we are looking forward to extract other requirement patterns according to the context of the problem in hand and provide support for them in our methodology. A plug-in retrieving documents from ProR and the Rodin editor can help to implement and automatically use our patterns.

[5]plug-in of the Rodin platform available in http://wiki.event-b.org/index.php/Event-B_Statemachines

[6]http://wiki.event-b.org/index.php/Project_Diagram

References

[1] Jean-Raymond Abrial. Formal Methods in Industry : Achievements, Problems, Future. In *28th International Conference on Software Engineering, Shanghai, China*, pages 761–768, 2006.

[2] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[3] Frédéric Boniol and Virginie Wiels. Landing Gear System case Study. In *ABZ Conference, Communications in Computer and Information Science, Springer*, volume 433, pages 1–18, 2014.

[4] EEIG ERTMS Users Group. Hybrid ERTMS/ETCS Level 3. Technical report, 2017.

[5] Marc Frappier, Frédéric Gervais, Régine Laleau, and Jérémy Milhau. Refinement patterns for ASTDs. *Formal Aspects of Computer*, 26(5):919–941, 2014.

[6] Florian Galinier, Jean-Michel Bruel, Sophie Ebersold, and Bertrand Meyer. Seamless Integration of Multirequirements in Complex Systems. In *IEEE 25th International Requirements Engineering Conference Workshops, RE Workshops, Lisbon, Portugal*, pages 21–25, 2017.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[8] Fahad R. Golra, Fabien Dagnat, Jeanine Souquières, Imen Sayar, and Sylvain Guerin. Bridging the Gap between Informal Requirements and Formal Specifications Using Model Federation. In *International Conference on Software Engineering and Formal Methods, France*, pages 44–69, 2018.

[9] Thai Son Hoang, Andreas Fürst, and Jean-Raymond Abrial. Event-B Patterns and their Tool Support. *Software and System Modeling*, 12(2):229–244, 2013.

[10] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580 & 583, 1969.

[11] Michael Jastram. ProR, an Open Source Platform for Requirements Engineering based RIF. *In Systems Engineering Infrastructure Conference, SEISCONF*, 2010.

[12] Michael Leuschel and Michael Butler. ProB: A Model Checker for B. In *International Symposium of Formal Methods Europe, Pisa, Italy*, volume 2805 of *LNCS*, pages 855–874. Springer, 2003.

[13] Atif Mashkoor. The hemodialysis machine case study. In *5th International Conference ABZ: Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 329–343, 2016.

[14] Christophe Ponsard, Robert Darimont, and Arnaud Michot. Combining Models, Diagrams and Tables for Efficient Requirements Engineering : Lessons Learned from the Industry. In *Actes du XXXIIIème Congrès INFORSID, Biarritz, France*, pages 235–250, 2015.

[15] Imen Sayar and Jeanine Souquières. Du cahier des charges à sa spécification. In *16 ème journées AFADL, Montpellier, France*, 2017.

[16] Imen Sayar and Jeanine Souquières. La validation dans les premières étapes du processus de dévelopement. *Revue ISI-DAT, numéro spécial Décisions, argumentation et tracabilité dans l'Ingénierie des Systèmes d'Information*, 22(4):11–41, 2017.

[17] Axel van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.