Bridging the gap between informal requirements and formal specifications using model federation

Fahad R. Golra[1], Fabien Dagnat[2], Jeanine Souquières[1], Imen Sayar[1], and Sylvain Guerin[3]

[1] Université de Lorraine, CNRS, LORIA, F-54000 Nancy, France
[2] IMT Atlantique, IRISA, Université Bretagne Loire, F-29238 Brest, France
[3] Openflexo, 29280 Plouzané, France

Abstract. Software development projects seeking a high level of accuracy reach out to formal methods as early as the requirements engineering phase. However the client perspective of the future system is presented in an informal requirements document. The gap between the formal and informal approaches (and the artifacts used and produced by them) adds further complexity to an already rigorous task of software development. Our goal is to bridge this gap through a fine-grained level of traceability between the client-side informal requirements document to the developer-side formal specifications using a semi-formal modeling technique, model federation. Such a level of traceability can be exploited by the requirements engineering process for performing different actions that involve either or both these informal and formal artifacts. The effort and time consumed in developing such a level of traceability pays back in the later phases of a development project. For example, one can accurately narrow down the requirements responsible for an inconsistency in proof obligations during the analysis phase. We illustrate our approach using a running example from a landing gear system case study.

1 Introduction

General software development methods do not lend themselves to the kind of rigorous analysis necessary for ensuring the degree of assurance required for safety (or life) critical systems [1]. Formal methods attain that level of quality through proper documentation and significant analysis. In projects using formal methods, we usually come across domain artifacts (feasibility reports, existing models and software, standards, etc.), user requirements document (also serves as a frame of reference for the agreement between client and supplier) and the specifications document (used for formally defining the requirements). The specification documents are prepared to concretize the software development team's perspective of the software under development [2]. Where requirements document is an informal description of the system, a specification document uses rigorous formal methods that serve for verification and prototyping of a designed system.

As a development project progresses, artifacts contributing to its goal are produced. Traceability is the ability to link these artifacts together, so that one

can identify the relationship between them and trace back/forward to them. Due to the gap between the informal and formal approaches, a requirement is taken as a single unit of reference for traceability [3]. This amounts to a coarse-grained traceability that overlooks individual concepts in each requirement [4]. We argue that a fine-grained level of traceability that can link individual concepts in formal specifications to the individual concepts of informal requirements shall help reduce this gap. Different approaches propose using a controlled natural language [5] to solve this issue. Even though the use of controlled natural language helps reduce requirements ambiguity, it hardly offers any support for improving the level of traceability between the requirements and specifications.

In a previous work, we proposed a concept-level traceability between the informal requirements and the formal specifications [6]. As an extension to that work, we use semi-formal models to formalize this traceability mechanism. We chose model federation [7] to realize a framework for the development and co-evolution of requirements and specifications. Model federation is an approach that enables binding a set of models from heterogeneous paradigms together. This binding is defined through a behavior that specifies the evolution of federated models in the development of a software system. Using this approach we developed an open source tool that can link requirements documented in various formats (word processors, spreadsheets, databases, xml files or Reqif supporting tools) to the formal specifications. We illustrate the use of our framework using examples from the landing gear system case study [8].

The rest of this paper is organized as follows. First, we describe the gap between the informal and formal approaches in Section 2. The model federation approach in presented in Section 3. In Section 4, we explain the structural core and in Section 5, we describe the methodological aspects of our framework. We share the lessons learned in various case studies, in Section 6. Then in Section 7 we discuss the state of the art. Finally, we conclude this paper in Section 8.

2 The gap

Formal methods serve as the backbone of software engineering for critical and complex systems [9]. They guarantee the correctness of the system under development and help in early validation/verification of requirements. For example, Event-B [10] is a formal method for modeling and reasoning about large reactive and distributed systems. It is centered on the notion of transitions. Models are developed using two basic constructs: contexts and machines. Building a specification is a gradual process that uses context extension and machine refinement. Rodin [11], an Eclipse based IDE for Event-B provides effective support for refinement and mathematical proofs.

When working on a critical and complex system, one has to deal with both informal and formal models during requirements engineering and/or early architecture design. The most obvious informal model is the requirements document that lists the requirements of the system to be built using natural language. Even though some approaches propose a controlled structure for writing

the requirements [5], the requirements document remains informal. Tools like spreadsheets and word processors are still extensively used for maintaining the requirements [12]. Such tools along with other requirements management tools (*e.g.* Rational DOORS[4]) may provide the necessary flexibility for requirements management, but they offer little support when it comes to traceability. Other tools specifically designed for requirements traceability (*e.g.* Reqtify[5]) offer a very coarse-grained traceability. They consider a requirement as a unit concept and link it to other artifacts of software development. This makes it hard to keep track of individual concepts that form the core of a system design.

Unless the traceability approaches can pin down the concepts that lead to problems in specifications (*e.g.* conflicting requirements), the problem of gap between the informal and formal approaches in early software development can not be resolved. Imposing formal languages for documenting requirements is not possible because requirements elicitation is often a shared responsibility of clients and suppliers. In an earlier work, we also shared the case where the requirements document was almost completely prepared by the client organization [13]. Sometimes, the clients of critical and complex industry, especially from aviation and defense sectors, prepare the requirements documents in advance and then call for an open bidding for the projects. In requirement engineering, especially for critical and complex systems, where formal specification helps ensuring the correctness of the system, these informal requirements become the Achilles heel of the complete process. Coarse-grained traceability to the requirements level is not sufficient enough to cover this gap of informal to formal models. We argue that a very fine-grained level of forward and backward traceability to/from the specifications can reduce this gap. With such an approach at hand, one can even look forward to semi-automatic co-evolution of requirements and specifications.

3 Model federation

Model Federation is a modeling approach where the focus is shifted from models to group of models. Instead of manipulating a single large model, it promotes using a set of interdependent models. This approach stems from the fact that a model is not an isolated entity, rather it depends on other models. For example one can federate a document file (.docx) with a list of states and a minimalistic state automaton (xml file format of UPPAAL). A model federation is therefore a chosen group of models reifying their dependencies to serve an intention. In the context of the automaton federation example , it aims at ensuring the consistency of the textual list of states and the automaton . An action on a member of a federation might impact the whole federation. Hence, each action on a model must be considered as an action on the federation. For instance, changing the name of a state is an action both on the document file and the automaton.

While this conceptual approach can be applied in any discipline using models, it is at its best when dealing with heterogeneous models pertaining to different

[4] `https://www.ibm.com/us-en/marketplace/rational-doors`

[5] `https://www.3ds.com/products-services/catia/products/reqtify/`

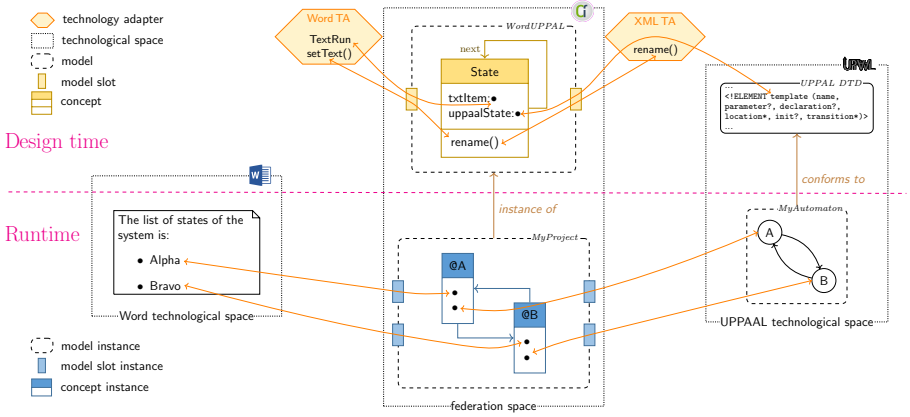

Fig. 1. An example of a model federation

paradigms. Co-evolution is difficult in such a scenario [14], especially if one wants to keep the various members of the federation in their respective paradigms [7]. It is often easier to act directly on the federation rather than acting on it members first and then recovering the consistency. The role of the federation therefore is to reify the process of ensuring consistency between the models of the federation. Notice that the level of consistency can vary, depending on the intention of its designer. Some federations may constrain the possible actions on the models of the federation to ensure a strict level of consistency. While others may choose to gather the inconsistencies and require human intervention to resolve them.

The approach is relatively young but we have designed a framework with associated methods and tools. This framework relies on the following architecture. A federation gathers a set of conceptual models, named *virtual models* and a set of *federated models*. Each federated model pertains to a *technological space* and uses the language of its specific paradigm while a virtual model is built using the Federation Modeling Language (FML). Each federated model can be viewed as an autonomous component while the virtual models serve as control components. Figure 1 presents a simplified version of the automaton federation example that groups these different models. The upper half of the figure illustrates the design of a federation with a model coming from the Word technological space (.docx file), another coming from the XML technological space (UPPAAL file) and one virtual model reifying the dependencies between the textual data and the corresponding automaton. A *technology adapter* (TA) is a reusable library that defines connections between the FML execution engine and a particular technological space.The model federation framework provides ways to define TA. The *automaton* virtual model, shown in Figure 1 relies on the two technology adapters (Docx TA and XML TA) for accessing the models of respective technological spaces.

FML is a language designed to define virtual models. A virtual model is composed of a set of *concepts*, while itself being a concept. Hence, virtual model are structuring units while concepts are the core entities. A concept has a set of *roles* and *behaviors*. A parallel to object-oriented approach can be useful to

understand FML[6]. A concept corresponds to a class, its roles to the attributes of the class and its behaviors to the methods of the class. In our example, a concept State is presented with two roles, txtItem and uppaalState. These roles have types defining the kind of value the role will point to at runtime. Our example illustrates three forms of such types *i.e.* another concept for next , a type defined in a TA for txtItem or a type defined in an external model for uppaalState . Whenever a type external to the federation space (from a TA or an external model) is used, one needs to use a *model slot*. A model slot is a mediation entity, associated with a TA, in charge of giving access to external elements of the corresponding technological space. A model slot defines a view on an external model by interpreting it as a set external concepts. Notice that a model slot can limit its interpretation to the needed part of the external model.

FML is designed to define not only the structure of the virtual models but also their behavior. Beware, here the behavior means the collection of actions an engineer will be able to perform on a model federation. It is different from the behavior of the System Under Study. The rename operation already cited for the automaton federation is an example of this behavior. One could also define operations to add or remove states, transitions , *etc.* The renaming operation is defined by a rename behavior in the concept State. It uses the setText action of the Word TA and rename from the XML TA. When the FML execution engine runs a federation, it creates virtual model instances containing concept instances. Some concept instances are connected to external elements through model slot instances. The lower part of the figure 1 illustrates this runtime phase.

The tool support for model federation framework, Openflexo[7], is developed as an open source initiative with active community around it. This tool offers a FML execution engine with an interactive virtual model design environment. It has been used in several use-cases including model mapping, multi-paradigm process modeling and enterprise architecting. It has also been used to build a tool, the freemodeling editor that has been put to practice in industrial projects [15]. As of today, this tool offers some mature technology adapters (*docx* and *excel* for documents, EMF and OWL for modeling languages, JDBC for databases, REST and XMLRPC for external services and one for diagramming tools) and some other rudimentary ones (pdf, http, XML and powerpoint).

4 Linking requirements to specifications

In order to overcome the informal to formal barrier, as described in Section 2, we developed an approach to link requirements and formal specifications using a fine-grained level of traceability. We develop a model federation using three virtual models in federation space *i.e.* requirements, specification and glossary virtual models. For the requirements, this federation connects to any of the three technological spaces shown in Figure 2. It may connect to other technological

[6]Beware, even though useful for comprehension, this correspondence is not reliable, as some aspects of FML do not map to object oriented concepts

[7]`http://openflexo.org` and `https://github.com/openflexo-team`

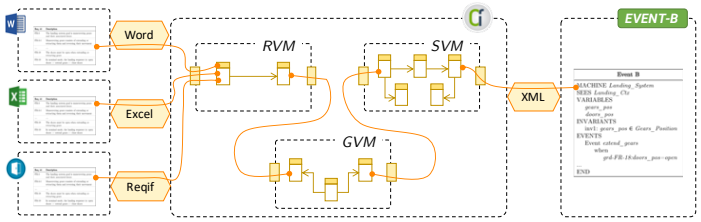

Fig. 2. Model federation for requirements to specification tracing

spaces for which the tool offers a technology adapter *e.g.* databases, EMF, service oriented platforms, *etc.* For formal specifications, currently we are supporting Event-B technological space through XML technology adapter.

4.1 Requirement Virtual Model

We benefit from the strength of model federation by developing a virtual model for requirements. Instead of rewriting the requirements using a formal grammar or transforming the requirements into another (formal or semi-formal) model, the *requirement virtual model* interprets the textual requirements for our specific use of traceability. It allows identifying and tracing back to individual concepts within the textual description. For our specific use, we only need two concepts from a requirement *i.e.* the requirement identifier and its textual description. Functional and non-functional requirements are treated the same way, as long as there is a corresponding concept in both the requirement and the specification. Using existing *Technology Adapters*, requirement virtual model can connect to heterogeneous platforms to get any requirements from a word processor, spreadsheet or RIF/ReqIF compliant XML formats (*e.g.* DOORS, ProR, etc.).

`Requirement` concept of requirement virtual model, as illustrated in Figure 3 gets the identifier and the textual description of the requirement from the connected *Requirements Technological Space*. It also contains two boolean type roles `isValidated` and `isConsidered` that are used to relay back the information to the concerned stakeholders about the status of a requirement; whether or not it was included in the specification and does it pose any proof obligation issue. The `IdentifiedConcept` refers to a word/sub-phrase in the textual description of the requirements document which carries an equivalent formal concept in the system specification. The exploration and selection of concepts ensuring the conformity to the defined needs is a progressive activity carried out for the development of specifications in complex software development. For example, the European Cooperation for Space Standardization details a complete process in

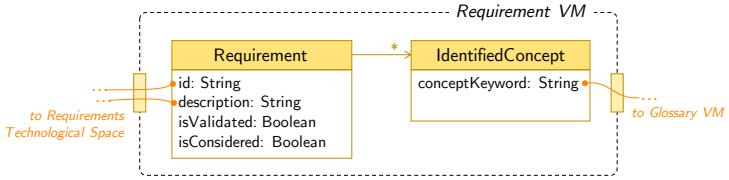

Fig. 3. Simplified requirement virtual model

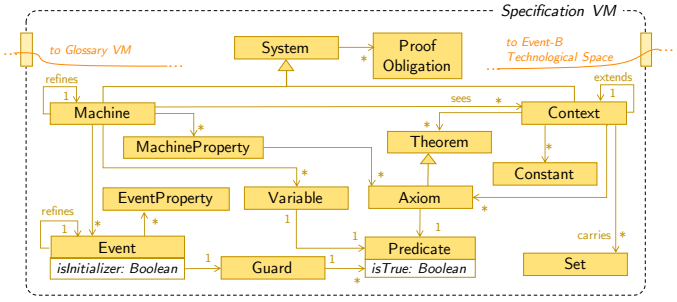

Fig. 4. Excerpt from the specification virtual model

ECSS-E-ST-10-06C, starting from the identification of possible concepts to the establishment of technical specifications [16]. The behavior part of this model, not shown in the figure, allows to describe the operations like relaying the information to the stakeholders, observing any change in the requirements, triggering certain behavior in the connected Glossary VM, etc.

4.2 Specification Virtual Model

Specification VM interprets a specification with the intention of linking it to the requirements. The long-term objective of this virtual model is to interpret specifications from different (lightweight) formal methods, however we have focused on the Event-B specifications for the moment. Still, the use of model federation offers tool independence by allowing us to use the same virtual model for *Atelier B*, *Rodin* or *B-Toolkit*. Figure 4 illustrates the key concepts of the specification virtual model. The notions of local variable, action, variants and invariants, *etc.* are abstracted behind `EventProperty` and `MachineProperty` for brevity.

The goal of interpreting an Event-B model is to gather the formal concepts from a specification and to identify the kind of those concepts. A formal concept can be of any kind *e.g.* a machine, variable, constant, etc. We believe that the behavior of a trace link between an informal concept and a formal concept depends on the kinds of those concepts. For example, when an informal concept in a requirement is traced to a constant concept of a specification, one can define the behavior of the trace link such that a change in an informal concept automatically changes the formal concept, but a change in formal concept requires a validation of a requirement engineer to be propagated to the informal concept. For such federation level behavior, the user can add an operation in Specification VM that triggers the behavior of the linked virtual models. Specification VM already contains the behavior for observing the Event-B technological Space and triggering various behaviors of the Glossary VM.

4.3 Glossary Virtual Model

Glossary VM is the key model that binds *Requirement VM* and *Specification VM*. The `InformalConcept` from this model is linked to the Requirement VM,

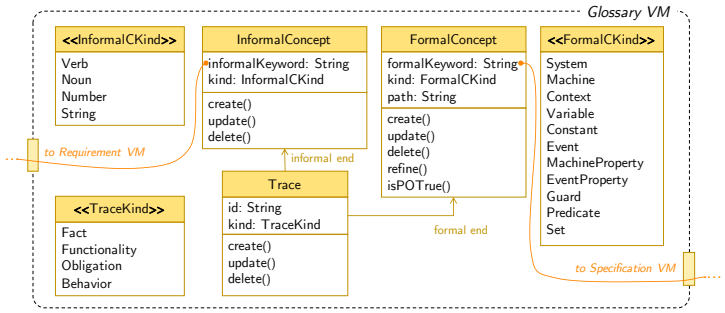

Fig. 5. Glossary virtual model

from where it gets the informal keyword of the concept. This keyword can be of any kind specified by the `InformalCKind`. For the moment, we have left it to the user to choose the kind of the concept, but for an industrial application, one can integrate Natural Language Processing techniques (*e.g.* [17]) that can parse, extract and categorize the concepts from the requirement descriptions.

The `FormalConcept` of the glossary virtual model is linked to the *Specification VM*, from where it gets the `formalKeyword`. It also specifies the kind of the concept amongst the ones described by the `FormalCKind`. Where `InformalConcept` only defines the basic manipulation behaviors like `create`, `update` and `delete`, the `FormalConcept` also defines the `refine` and `isPOTrue` behaviors. The `refine` behavior carries the information about the refinement of an Event-B machine to the corresponding requirement in the requirements document. The `isPOTrue` behavior notifies the concerned stakeholders of the requirement, whether or not the proof obligations of the corresponding Event-B machine are validated. `Trace` binds the `InformalConcept` with the `FormalConcept`. Each *trace* has a unique identifier. The `create` and `update` behaviors assign a kind to the trace amongst the ones defined by `TraceKind` :

- A **Fact** connects an informal concept of `Noun`, `Number` or `String` kind to a formal concept of `Constant`, `Set`, `Variable`, etc. These trace links are used by requirements that specify facts about the future system.
- A **Functionality** connects an informal concept of `Verb` kind to a formal concept of `Event` kind. A requirement linked to this kind of trace describes an expected functionality of the system under development.
- An **Obligation** connects an informal concept of `String` kind to a formal concept of `Axiom`, `Invariant`, `Theorem` and `Guard` kinds. These trace links are used for preconditions or postconditions on the functioning of a system.
- A **Behavior** is a collection of *Functionality* links, such that their events must be performed in a temporal order. The informal concept is a `String` (a sub phrase) that contains the functionality concepts along with their order and links them to each of their corresponding formal concepts. The informal concept in the requirement specifies a behavior expected of the future system.

It is important to note here that our objective is not to exhaustively define the behavior of all kinds of trace links, but to propose a mechanism where such

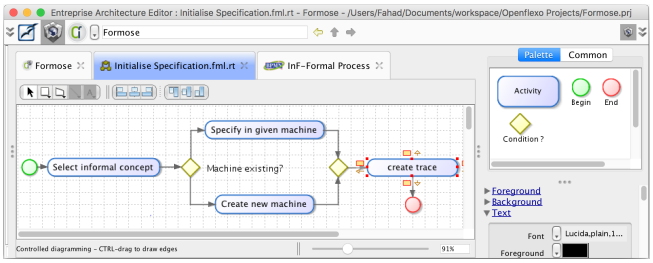

Fig. 6. Enterprise architecture editor

behavior can be specified. The three virtual models *i.e.* RVM, SVM and GVM, are used by the tool providers. An end user defines its traceability links with a set of available behavioral templates provided by the tool provider. The virtual models are instantiated in the background. Unless the user needs a customized behavior, she does not need to work with the virtual models. A strong motivation for choosing model federation was its ability to decouple the technological dependence from the core behavior of the methodology. For example, the behavior chosen for a trace link between a requirement and Event-B machine does not depend on the tool used for documenting the requirements. The technological aspects of a requirement coming from DOORS or from an Excel spreadsheet do not alter this behavior. They are confined in the Technology Adapter.

5 Process driven approach

To witness the full potential of our framework, one needs to define the process that integrates the building blocks from the models presented in the previous section. Using the Openflexo tool, we have developed a process modeling editor that serves for defining the process, as shown in Figure 6. Once defined, each activity is linked with the behaviors of individual virtual models for the execution. The user has the liberty to define a desired behavior for each activity.

As the tool follows a process driven approach, it does not impose a specific behavior to the user. We demonstrate the use of our framework by discussing few examples of the activities from the landing gear system case study [8].

1. *Initializing a machine:* At the start of the specification process, there exists no corresponding Event-B machine for the concepts identified in the requirements document. The user identifies an informal concept, for which (s)he plans to develop an Event-B machine. Same thing happens, when a concept found in the requirements document does not correspond to any existing machine and the development of a new machine is intended. In the example of Figure 7, the user identifies "landing system" as informal concept. As per the process of Figure 6, when no corresponding machine exists, the framework offers to create a new one. *Specification VM* can be used for creating a stub for the new machine. Once the machine is created, "*Landing_system*" is identified as a machine concept of *Specification VM* and it is linked to the informal concept "landing system".

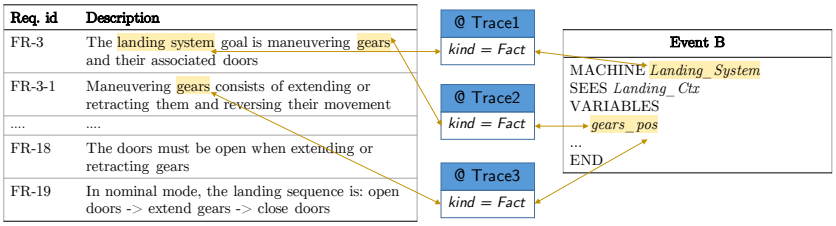

Fig. 7. Virtual Model Instance - Initialization[8]

2. *Updating a trace link:* The default behavior of the framework when the name of an Event-B machine (*e.g. Landing_System*) changes is to update the glossary and maintain the trace link. Conversely, when the name of the informal concept *landing system* changes, the framework generates a notification for the system analyst to check for probable inconsistencies. Even though the framework provides a default behavior for the activities associated with tracing, thanks to the model federation approach, the user has the flexibility to define a different behavior. Usually the behavior of such activities is encoded within the tool implementations, and changing the behavior is impossible or at least difficult in case of open source software. For model federation, this behavior is not coded in the tool, rather defined in the model itself.
3. *Adding new trace links:* The process of linking requirements to formal specifications is fairly flexible and varies from case to case. One can choose an informal concept and link it to an existing formal concept. But it can also go the other way round, when one selects an existing formal concept and links it to a concept from a requirement. In Figure 7, we see that "gears" from FR-3 is linked to the "*gears_pos*" in *Landing_System* machine. Once this link is formed, the user goes in the reverse direction for linking this formal concept to other identified concepts from the requirements that refer to it. Similarly, the user links the informal concepts "extending" and "extend gears" to a corresponding formal concept "*extend_gears*" through a trace link of type functionality, as shown in Figure 8. It is important to know that multiple derivatives of a single informal concept (*e.g.* extending, extend gears, etc.) do not need to have different corresponding formal concepts. One can notice in the figure that the complete description of FR-18 forms an informal concept linked to the guard of the *Landing_System* machine.
4. *Early requirements validation:* The process of tracing the requirements to the formal specifications forms the basis of validation. A requirement can have trace links to multiple Event-B machines and similarly a machine can have links to multiple requirements. The process of validation does not need to wait till the requirements document is complete. Hence, as the requirement document is under development, the formal specification process can start in parallel. This way requirements are traced to the specifications as and when they arrive. The `isPOTrue` property of the formal concept in the *Glossary VM* keeps track of the proof obligations of the linked Event-B machine. If the

[8] This figure does not show the instances of RVM and SVM for the reason of brevity.

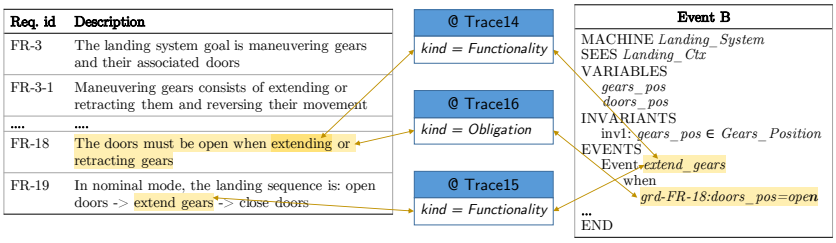

Fig. 8. Virtual Model Instance - Traces[8]

proof obligations of all the machines traced by a requirement's concepts are proved, the `isValidated` property of the requirement virtual model returns *true*. If a requirement is linked to the glossary but the trace is not complete to the formal specification, the `isConsidered` property of the requirement virtual model return *false*. This way the information about the validity of the requirements is relayed back to the stakeholders.

5. *Refinement of requirements and specifications:* Requirement elicitation is a gradual process that involves the refinement of abstract level requirements to the concrete level requirements. This process can be carried out at requirements or specifications. When an informal requirement is refined, all the informal concepts reappearing in the refined requirements with their corresponding formal concepts are notified to the stakeholder for possibly new trace links. In case of a refinement of an Event-B machine, the information is passed through the `refine` property of the glossary virtual model. The goal is to trigger new trace links from the requirements to the new machine. Update 27

The tool implementation for this approach also supports requirements elicitation, links to domain models, ontologies and goal models, *etc.* but they are out of scope for this article [9]. The intention is to show parts of the default process and the possibility to define user-specific processes. The tool being modular, we have linked this *requirement to specification traceability module* with our previous modules e.g. goal modeling and requirements elicitation modules [13].

6 Lessons learned

During the early development of the approach, we focused on two case studies *i.e.* the landing gear system [8] and the hemodialysis machine [18]. Finally we implemented our approach on a real-life case study provided by our industrial partner from aviation and aerospace industry, under a research project, FORMOSE, funded by French National Research Agency (ANR)[10]. Based on our experiences with these implementations, we share the following lessons learned:

1. *Parallel incremental development of requirements and specifications:* During our case studies, we found out that it is not necessary to wait for the require-

[9] FORMOD tool is available at `https://downloads.openflexo.org/Formose`

[10] Bound by a non-disclosure agreement, we can't share the details of this case study.

Req. id	Description
FR-3	The [Landing_System] goal is maneuvering [gears_pos] and their associated [doors_pos]
FR-3-1	Maneuvering [gears_pos] consists of [extend_gears] or [retract_gears] and reversing their movement
...	...
FR-18	The [doors_pos] must be [open_doors] when [extend_gears] or [retract_gears]
FR-19	In nominal mode, the landing sequence is: [open_doors] → [extend_gears] → [close_doors]

Fig. 9. Specification version of requirements document

ments document to start the specifications. As soon as requirements start to pour in, the development of specifications can start. The parallel and incremental development of both these artifacts helps requirements elicitation.

2. *Fine-grained traceability:* We applied a very fine-grained level of traceability going down to the concept level, within each requirement. Such a level of traceability helped the analysts to associate proper requirements to the justifications of each concept at the specification level. It also helped in categorizing the requirements according to their corresponding implementations in the formal specifications. Prioritization of requirements from the clients perspective is a common practice, but when combined with the specification view of priorities, it helps stakeholders to take informed decisions.
3. *The validation of the informal requirements:* Late validation of requirements after the requirements engineering phase increases the cost of corrective measures. The proposed framework helped in bringing the validation process close to requirements elicitation so much that the validation of requirements is done in parallel with the requirements document development. Once developed, it helps in proving the correctness of the formal specification in relation to the concepts present in the informal requirements, all along the development process. Maintaining the trace links in a glossary reduces the effort of validating complex specification models through refinement.
4. *Automation of traces:* One of the main reasons for using model federation for the linking informal requirements to formal specifications was the possibility of operationalizing the trace links. A trace link is not just a pointer from an informal concept to a formal concept, it contains a behavior. This allows to (semi-)automate some tasks of linking the two artifacts. Some behaviors we came across during the case studies were automatic update of constants, triggering notifications to the stakeholders, generating requests for proof obligations, changing the state of requirements from valid to conflicting, etc.
5. *Verification of requirements:* Because the traceability of requirements to formal specification was taken to a fine-grained level, the introduction of corresponding concepts helped in the verification of formal specifications. We found out that such traceability helps in detecting omissions in the requirements (initial states, implicit undescribed requirements, or absence of scenarios) or contradictions between the specifications and requirements.
6. *Specification versions of clients documents:* An interesting finding of the implementation of our framework to the industrial project was that the specification stakeholders relate more with the formal text than the informal one.

The available trace links made it very easy to generate a version of requirements document where the informal concepts were replaced with the formal ones, as shown in Figure 9. This eased their comprehension of requirements and improved the communication within the team.

7 Related work

The gap between the informal requirements and the formal specifications was acknowledged and has been a topic of research interest for the past three decades. Deriving VDM specifications from Structural Analysis (mostly Data Flow Diagrams) [19], Object Constraint Language (OCL) specifications from UML use cases [3] and system specifications from Problem Frame descriptions [4] are few of the examples to bridge this gap. However, most of such efforts are tools and technology specific endeavors. They lack a generic methodology that can be applied in a variety of configurations. The main focus of our work is to reduce the technology dependence from the methodology of linking informal requirements to formal specifications. From the requirements perspective, we accept requirements coming from any tool or described in any formalism. For the specifications, we only implemented Event-B model for now. This is an implementation shortcoming but it does not alter the proposed methodology.

KAOS is a refinement-based goal-oriented methodology for deriving specifications formalized using Linear Temporal Logic from informal requirements [20]. KAOS facilitates the derivation of system specifications through refinements, but imposes its own requirements description (goal modeling) methodology and constrains the language used for formal specifications. Besides, KAOS suffers from the lack of support for non-functional requirements. Li et al. [21] present another refinement based approach for the transformation of informal requirements to formal specifications, that can handle NFRs. They use a requirement ontology for classification and propose a requirements modeling language. The advantage of the proposed methodology is that the user is not restricted to any specific requirements specification method or tool.

Our work is notably closest to the approach of Jastram et al. where their requirement model differentiates between phenomena (state space and transitions of the system) and artifacts (the restriction on states and transitions) [22]. They classify the artifacts into Domain Knowledge (W), Requirements (R), Specifications (S),Program (P) and Programming Platform (M). Once formalized, these elements of requirements are mapped to Event-B, using ProR [23]. The main difference with our approach is that we propose explicit definition of traceability behavior in the trace links. Apart from mapping the concepts of informal requirements to formal specifications, we classify different kinds of trace links. The behavior of each kind of trace link is then reused for providing semi-automatic co-evolution of requirements and formal specifications.

Heisel et al. proposed a requirements elicitation process that is independent of the specification language [24]. An extension to their work using Rodin and ProR offers a fine-grained traceability between informal requirements and formal

specifications [6]. We have proposed yet another extension to this work using model federation. The advantage of this extension is that the trace links are now fine-grained to a concept level and that they contain the behavior of the trace, rendering them executable. The main limitation of our approach is the cost it incurs. Indeed, the process of maintaining the consistency between the requirements and the formal specifications must be reified. A method engineer has to define the corresponding behavior. Furthermore, the specification engineer needs to invest time for linking individual concepts of informal requirements to the formal specifications. However, this investment provides improved clarity and the possibility of automating certain activities in case of requirements change. Notice also that probably common behaviors would emerge and provide a set of reusable federation behaviors, lowering the cost of their design.

8 Conclusion and future work

We proposed a framework for linking informal requirements to formal requirements specifications. The main contributions of our approach are: (i) fine-grained traceability between individual concepts in a requirement and individual concepts in a formal specification, (ii) a mechanism for the incorporation of behavior within the trace links, and (iii) tooling and methodology for the incremental development and co-evolution of requirements and formal specifications. We are currently working on an operational semantics of FML using a process calculus encoding similar to [25]. The aim of this semantics is twofold. First, we plan to check the correctness of our FML interpreter. Second, we would be able to prove properties about behaviors, for example, a modification of an artifact leads to a corresponding modification of artifacts depending on it. Proving properties of the process alongside the product would help certification of critical systems.

References

1. Coram, M., Bohner, S.: The impact of agile methods on software project management. In: 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS., IEEE (2005) 363–370
2. Clark, R.G., Moreira, A.M.: Formal specifications of user requirements. Automated Software Engineering **6**(3) (1999) 217–232
3. Giese, M., Heldal, R.: From informal to formal specifications in UML. In: Intern. Conf. on the Unified Modeling Language, Springer (2004) 197–211
4. Seater, R., Jackson, D., Gheyi, R.: Requirement progression in problem frames: deriving specifications from requirements. Requirements Engineering **12**(2) (2007) 77–102
5. Mavin, A., Wilkinson, P., Harwood, A., Novak, M.: Easy approach to requirements syntax (EARS). In: Inter. Requirements Engineering Conf., IEEE (2009) 317–322
6. Sayar, I., Souquières, J.: La validation dans les premires étapes du processus de développement. ISI-DAT **22**(4) (2017) 11–41
7. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Addressing modularity for heterogeneous multi-model systems using model federation. In: Companion Proc. of the Intern. Conf. on Modularity, ACM (2016) 206–211

8. Boniol, F., Wiels, V.: The landing gear system case study. In: Intern. Conf. on Abstract State Machines, Alloy, B, TLA, VDM, and Z, Springer (2014) 1–18
9. Rierson, L.: Developing safety-critical software: a practical guide for aviation software and DO-178C compliance. CRC Press (2017)
10. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
11. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. International journal on software tools for technology transfer **12**(6) (2010) 447–466
12. Behutiye, W., Karhapää, P., Costal, D., Oivo, M., Franch, X.: Non-functional requirements documentation in agile software development: Challenges and solution proposal. In: Product-Focused Software Process Improvement, Springer (2017) 515–522
13. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Continuous requirements engineering using model federation. In: 24th International Requirements Engineering Conference (RE). (Sept 2016) 347–352
14. Hebig, R., Giese, H., Stallmann, F., Seibel, A.: On the Complex Nature of MDE Evolution. In: Model-Driven Engineering Languages and Systems. Volume 8107 of LNCS. Springer Berlin Heidelberg (2013) 436–453
15. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Using free modeling as an agile method for developing domain specific modeling languages. In: Proc. of the ACM/IEEE 19th International Conf. on Model Driven Engineering Languages and Systems. MODELS '16, New York, USA, ACM (2016) 24–34
16. ECSS: Space Engineering - Technical Requirements Specification. Standard ECSS-E-ST-10-06C, European Cooperation for Space Standardization (2009)
17. Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F.: Automated checking of conformance to requirements templates using natural language processing. IEEE transactions on Software Engineering **41**(10) (2015) 944–968
18. Mashkoor, A.: The hemodialysis machine case study. In: Intern. Conf. on Abstract State Machines, Alloy, B, TLA, VDM, and Z, Springer (2016) 329–343
19. Fraser, M.D., Kumar, K., Vaishnavi, V.K.: Informal and formal requirements specification languages: bridging the gap. IEEE transactions on Software Engineering **17**(5) (1991) 454–466
20. Van Lamsweerde, A.: Requirements engineering: From system goals to UML models to software. Volume 10. Chichester, UK: John Wiley & Sons (2009)
21. Li, F.L., Horkoff, J., Borgida, A., Guizzardi, G., Liu, L., Mylopoulos, J.: From stakeholder requirements to formal specifications through refinement. In: International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer (2015) 164–180
22. Jastram, M., Hallerstede, S., Leuschel, M., Russo, A.G.: An approach of requirements tracing in formal refinement. In: International Conference on Verified Software: Theories, Tools, and Experiments, Springer (2010) 97–111
23. Jastram, M.: ProR, an Open Source Platform for Requirements Engineering based RIF. In: Systems Engineering Infrastructure Conference, SEISCONF. (2010)
24. Heisel, M., Souquières, J.: A method for requirements elicitation and formal specification. In: Intern. Conf. on Conceptual Modeling, Springer (1999) 309–325
25. Wong, P.Y., Gibbons, J.: Formalisations and applications of BPMN. Science of Computer Programming **76**(8) (2011) 633 – 650