A DSL for requirements in the context of a seamless approach

Florian Galinier

A DSL for Requirements in the Context of a Seamless Approach

Florian Galinier
IRIT, Toulouse University
Toulouse, France
florian.galinier@irit.fr

ABSTRACT

Reducing the lack of consistency between requirements and the system that should satisfy these requirements is one of the major issue in Requirement Engineering (RE). The objective of my thesis work is to propose a seamless approach, allowing users to express requirements, specifications and the system itself in a unique language.

The purpose of formal approaches is to reduce inconsistency. However, most developers are not familiar with these approaches, and they are not often used outside the critical systems domain. Since we want that non-experts can also use our approach to validate systems in the early stage of their development, we propose a Domain Specific Language (DSL) that is: (i) close to natural language, and (ii) based on a formal semantics. Using Model-Driven Engineering (MDE), this language bridges the gap not only between the several stakeholders that can be involved in a project, considering their different backgrounds, but also between the requirements and the code.

KEYWORDS

Requirements engineering, DSL, Seamless development, Traceability, Verification and Validation

1 RESEARCH PROBLEM

One of the main challenge in Requirements Engineering (RE) is to introduce formality in the expression of requirements. If formal approaches are used in critical systems, most of the time requirements are still expressed in Natural Language (NL). This can be explained by the force of habits, by the lack of knowledge on formal methods, or simply by the need to use a language understandable by all of the stakeholders. However, the use of a formal approach to express requirements shall lead to validate the systems in a rigorous way.

To overcome the difficulty of formal methods adoption, traceability is often used. This can help to detect which requirements are satisfied – providing a coverage information –, but given that traceability links are not semantically defined, these links cannot be automatically analyzed.

There is so a main question to address: *How to link requirements and other artifacts (such as requirements or even system parts) to automatically validate a system?* This question also raises the problem that in complex systems, stakeholders with different backgrounds are involved and often use heterogeneous tools. INCOSE [1] emphasizes this need to conciliate the several views of a system, and address it as a major challenge.

These questions are critical. Indeed, a lack of consistency between requirements and systems can lead to dramatic failures, such as some of the one listed in [2].

2 PREVIOUS WORKS

As we stated in [3], two worlds can be distinguished in RE[1]. Formal methods are, by definition, the more mathematically rigorous, and approaches like Event-B [4] or VDM [5] have been successfully used among years. As mentioned in section 1, the major issues of these approaches is linked to their main advantage: there are formal, and so, discouraging to non-experts.

That is why the most used tools, industrial ones, rely on Natural Language. IBM Rational Doors [6] or Reqtify [7] thus allow to create traceability links between requirements, expressed in a Microsoft Word document for example, with a part of the system – e.g., some C code. SysML [8] also proposes a requirements diagram, that allows users to link requirements to other parts of systems (such as blocks). If SysML's relationships own a type, contrary to the previously mentioned industrial approaches, there are not semantically defined.

Some approaches try to propose a bridge between the "formal world" and the "NL world". For example, Relax [9] or Stimulus [10] propose to express requirements in a constrained NL, allowing users to express requirements as they usually do. Their approaches are however semantically defined – in fuzzy branching temporal logic [11] for Relax and on a programming language based on Lucid Synchrone [12] and Lutin [13] for Stimulus. This helps the user to check requirements, while using an easy-to-handle tool. However, these approaches do not address the problem of linking requirements to the system, and are more specifically designed to ease and strengthen the requirements elicitation (making it rigorous).

[1] We are currently working on a survey of formal approaches for requirements.

https://doi.org/10.1145/3238147.3241538

The Single Model Principle proposed in [14] and adapted in [15] recommends the use of an unique paradigm to express the several artifacts of the system. This should help to avoid the gap introduced by the use of several languages, allowing users to validate the system while developing it. The proposed approach is based on this seamless idea.

3 PROPOSED APPROACH

3.1 Seamless Requirements

In [16], a set of patterns are proposed to transform NL requirements to a programmatic representation, based on Design by Contracts [17]. These representations – named *specification drivers* in [18] – shall be used to validate that other parts of system, controlled by these specification drivers, are complying with the requirements.

By using a programming language that integrates Design by Contracts (such as Eiffel or JML[19]), this approach is seamless. It allows users to represent requirements (via *specification drivers*) and system implementation (via the code) in a same paradigm. Moreover, the solver can be used to prove this validation (for example, the Autoproof tool [20] for Eiffel), by calling it on specification drivers.

```
-- Require an ambulance to be
mobilized_within_two_time_units
local
  old_distance: INTEGER
do
  from
    old_distance := distance
    occurs_allocate
  until
    mobilized = mobilize or
    (distance - old_distance) >= 2
  loop
    mobilize_ambulance
  end
ensure
  is_mobilized: mobilized = mobilize
  distance_less_than_two: distance -
      old_distance <= 2
end
```

Listing 1: Example of a functional requirement from the London Ambulance Service (LAS) system [21] expressed with specification driver in Eiffel

Listing 1 is an example of a specification driver. This driver controls the validation of the requirement *"After being allocated, an ambulance shall be mobilized within two time units"*. The `mobilize_ambulance` feature is the controlled one – i.e., the feature that should satisfy the requirement.

3.2 Semantics of Relationships

Since it is possible in Eiffel to express requirements and other artifacts, we propose to explicit relationships between these artifacts. We use for this purpose the Eiffel Information System (EIS) mechanism. This mechanism exploits the Eiffel notion of note (equivalent to Java annotations), that let developers put information in the form of:

<Notes> ::= 'note' <Note>+

<Note> ::= <Tag> ':' <String> <NEWLINE>

This mechanism allows the users of EiffelStudio (the main IDE for Eiffel) to create links between parts of code (features, classes or clusters) and other documents (such as Microsoft Word, PDF, website, …). If one of the endpoint of this link is modified, the IDE warn the user that a change occurred and he should probably take care of it.

To make explicit the relationships linked to requirements expressed through specification drivers, we modify this mechanism[2]. More fine grain are thus possible, allowing users to link parts of features (such as assertions, used to express the specifications' constraints). EIS links can also be used to link parts of code between themselves (for example a specification driver and the feature that should satisfy this driver), and these relationships are now typed.

These add should lead to clarify the several relationships existing between artifacts.

```
1  -- Require an ambulance to be
2  mobilized_within_two_time_units
3  note
4    EIS: "src=requirements.docx", "ref=1.6"
         , "type=trace"
5    EIS: "src=mobilize_ambulance", "dest=
         is_mobilized,
         distance_less_than_two", "type=
         verify"
6  local
7    old_distance: INTEGER
8  do
9    from
10     old_distance := distance
11     occurs_allocate
12   until
13     mobilized = mobilize or
14     (distance - old_distance) >= 2
15   loop
16     mobilize_ambulance
17   end
18 ensure
19   is_mobilized: mobilized = mobilize
20   distance_less_than_two: distance -
         old_distance <= 2
21 end
```

Listing 2: Example of EIS links on specification driver of Listing 1

In Listing 2, we add EIS links to our previous example. Relationships are thus clarified. Actually, example giving, the EIS note line 4 links the specification driver to a textual version of it, in a Microsoft Word document, referenced by the bookmark *1.6*. The note line 5 details the role of assertions `is_mobilized` and

[2] https://github.com/fgalinier/EiffelStudio

distance_less_than_two that are used to verify the validation of the specification driver by the feature mobilize_ambulance. Moreover, we add semantics to these links, defined in Table 1. The notation used in the following is:

- R_i is a requirement;
- r_i is the specification driver of the requirement R_i;
- f is an Eiffel feature (a method or an attribute);
- a is an assertion in Eiffel (a pre or postcondition, or an invariant)

Table 1: Types of EIS relationships and their semantics.

Relationships	Semantics
Trace	Link with no semantics
Refine	R_1 refines $R_2 \stackrel{\Delta}{=} r_1$ redefine r_2
Contains	R_1 contains $R_2 \stackrel{\Delta}{=}$ r_2 is called in $r_1 \wedge (\nexists\, r_3 : R_3 \mid r_2$ is called in $r_3)$
Copy	R_1 copies $R_2 \stackrel{\Delta}{=}$ r_1 body is a unique call to r_2
Derive	R_1 derives from $R_2 \stackrel{\Delta}{=}$ r_1 is called in r_2
Satisfy	f contributes to satisfy $R_1 \stackrel{\Delta}{=} f$ is called in r_1
Verify	a verifies $R_1 \stackrel{\Delta}{=} a$ is an assertion of r_1

These semantics can be used in two different ways:

- it should lead to a complete requirements validation – e.g., a requirement R_1 that contains requirements R_2 and R_3 will be validate thanks to this semantics only if both contained requirements are validate;
- by checking if the semantics of the relationships is respected, users can have feedback on the matching of what they intended to express and these relationships.

Thus, by adding semantics on links between requirements and artifacts, we get a more precise information on the validity of the system. Besides, we plan to explore the inverse relationships, to detect patterns that can be used to generate relationships between requirements. This can also help to detect relationships between requirements coming from several stakeholders.

3.3 Addressing the Several Stakeholders

Addressing the several stakeholders is a quite difficult problem, since they used several kinds of representations.

This problem is a well-known problem on Model Driven Engineering (MDE), and models transformations can be used to overcome these gaps between languages. Instead of defining one-to-one transformations between several languages, we propose to define a modeling language, that can be used as a pivot.

We called this language Requirement Specific Modeling Language (RSML)[3]. It is a DSL with a concrete syntax in a NL style (such as Stimulus or Relax), semantically defined in Eiffel.

In Fig. 1, is an example of a functional requirement expressed in RSML. Using patterns mentioned in section 3.1, this requirement

[3] https://gitlab.com/fgalinier/RSML

```
[1.1] An incident shall be resolved when an incident happened.
```

Figure 1: Example of a requirement from the LAS expressed in RSML

is transformed in an Eiffel representation (given Listing 3). Links between the specification driver and the automatically generated feature that should satisfy this requirement, are also added. In a similar way than Behavior Driven Development (BDD) [22], RSML should allow engineers to verify that the system specification is correct regarding to the requirements. However, contrary to BDD tools such as Cucumber[23], RSML provide a a formal representation of requirements, that can be used for static analysis of specification. This feature will be an entry point used by the engineer that will write the specification, allowing him to control that the code is correct.

```
-- 1.1
an_incident_shall_be_resolved
note
    EIS: "src=requirements.rsml", "ref=1.1",
         "dest=event_an_incident_happened,
        an_incident_is_resolved", "type=
        verify"
    EIS: "src=resolve_incident", "type=
        satisfy"
    Description: "[
        [1.1] An incident shall be resolved
            when an incident happened.
    ]"
require
    event_an_incident_happened:
        an_incident_happened
do
    resolve_incident
ensure
    an_incident_is_resolved: an_incident =
        resolved
end
```

Listing 3: RSML requirement from Fig. 1 translated in Eiffel

We plan to propose drivers from RSML to other used notation for requirements, such as SysML, KAOS[24] or even Microsoft Word documents. This should lead to reduce the gap between requirements, specifications and implementation. We also expect to use Autoproof to find some inconsistencies between requirements in an early stage, in a complementary way to model-checking approaches such as Stimulus.

4 EVALUATIONS

To evaluate the proposed approach, we are currently exploring different ways of implementation.

First we want to consider different activity domains. So, we applied the approach on two case studies, one is the embedded system of the LGS ([25]), and the other one is the reactive system LAS ([21]), already seen in this paper. We intend to apply it also on

an Information System, a banking system for example. It will so be clear that RSML can apply on a large panel of activity domains.

Secondly, through these applications, we will consider several types of requirements (e.g., timing constraints, temporal requirements, ...). For now, the enactment of this approach on the LGS and the LAS allowed us twice to highlight issues. In the first case we identified it in a set of LTL rules formalizing the LGS' temporal requirements ([16]). In the second case, the failure of the Autoproof session was linked to a misinterpretation of one of the timing LAS requirements. Supporting all these types of requirements, our approach could prove efficient to express reactive systems requirements.

Thirdly, we are currently implementing the approach on a prototype that will be test in the scope of a process involving several stakeholders. At first, we intend to propose a subject of practical classes to students, in the framework of RE course. The main idea is to split the class in three groups. We shall supply three case studies. Each of the groups will have a specific case study to be handled and so a set of requirements, expressed in a MSWord document. Every group will first supply its own RSML code, then propose the corresponding Eiffel code with the traceability links to the reference document and endly run it with Autoproof to check the validity of its system. We will so prove the usefulness of the approach for both novices and advanced stakeholders alike.

Finally and border line, we would also like to make an experimentation to see if from the RSML code of the LGS we can deduce a valid set of LTL constraints. Actually, we believe that if from an RSML system we can not only deduce Eiffel systems, benefiting from its powerful environment (EIS, Autoproof, ...), but also LTL formulae, there would be possible to obtain Event-B systems, and so on. We will so enforce the usefulness of RSML, being able to use it to exploit others formal verification languages and tools.

5 CONCLUSION AND PERSPECTIVE

We present in this paper some solutions to lead the users to formally express requirements without any specific knowledge while being able to validate them.

We propose for this purpose a seamless approach of development. It will reduce the gap between requirements and system, using a unique language to express both of them, Eiffel. To ease the analysis of the whole requirements, we define the semantics of relationships that exist between requirements and other artifacts. We also present RSML, a modeling language providing a canvas to express requirements in a syntax that is near from natural language and so, easy to handle. Since we automatically translate RSML requirements in Eiffel code, we are able to use an Eiffel solver to validate the provided system. This should also help users to detect errors in requirements or in the system that have to meet these requirements as early as possible. This is an ongoing work, and proposed solutions are still to improve – e.g., we are extending the syntax of RSML and adding new relationships.

The first experiments give us some encouraging results, and we plan to apply our approach to more complex case study (with more requirements), coming from diverse domains.

One of the major remaining work is the creation of bridges with other formalisms of requirements' modeling (e.g., SysML, KAOS, ...) to inscribe our approach in a model globalization context. This should allow the users to use their usual tools while benefiting from the advantages of our approach.

REFERENCES

[1] INCOSE. *SE Vision 2025*. 2014. http://www.incose.org/docs/default-source/aboutse/se-vision-2025.pdf.

[2] Matt Lake. Epic failures: 11 infamous software bugs | Computerworld, 2010.

[3] Florian Galinier, Jean-Michel Bruel, Sophie Ebersold, and Bertrand Meyer. Seamless integration of multirequirements in complex systems. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 21–25. IEEE, 2017.

[4] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[5] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer-Verlag, 1978.

[6] IBM Rational Doors. https://www.ibm.com/us-en/marketplace/requirements-management. Accessed: 2018-05-23.

[7] Dassault Systems Catia Reqtify. https://www.3ds.com/products-services/catia/products/reqtify. Accessed: 2018-05-23.

[8] Object Management Group (OMG). *OMG Systems Modeling Language (OMG SysML™), V1.0*. 2007. OMG Document Number: formal/2007-09-01 Standard document URL: http://www.omg.org/spec/SysML/1.0/PDF.

[9] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In *2009 17th IEEE International Requirements Engineering Conference*, pages 79–88, 2009.

[10] Bertrand Jeannet and Fabien Gaucher. Debugging real-time systems requirements: simulate the "what" before the "how". In *Embedded World Conference, Nürnberg, Germany*, 2015.

[11] Seong-ick Moon, Kwang H. Lee, and Doheon Lee. Fuzzy branching temporal logic. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):1045–1055, April 2004.

[12] Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet. A Conservative Extension of Synchronous Data-flow with State Machines. In *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, pages 173–182, New York, NY, USA, 2005. ACM.

[13] Pascal Raymond, Yvan Roux, and Erwan Jahier. Specifying and Executing Reactive Scenarios With Lutin. *Electronic Notes in Theoretical Computer Science*, 203(4):19–34, June 2008.

[14] Richard Paige and Jonathan Ostroff. The Single Model Principle. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, RE '01, pages 292–, Washington, DC, USA, 2001. IEEE Computer Society.

[15] Bertrand Meyer. Multirequirements. *Modelling and Quality in Requirements Engineering (Martin Glinz Festscrhift)*, 2013.

[16] Alexandr Naumchev, Bertrand Meyer, Manuel Mazzara, Florian Galinier, Jean-Michel Bruel, and Sophie Ebersold. Expressing and verifying embedded software requirements. *arXiv preprint arXiv:1710.02801*, 2017.

[17] Bertrand Meyer. Applying 'design by contract'. *Computer*, 25(10):40–51, October 1992.

[18] Alexandr Naumchev and Bertrand Meyer. Complete contracts through specification drivers. *arXiv:1602.04007 [cs]*, February 2016. arXiv: 1602.04007.

[19] Gary T Leavens, Albert L Baker, and Clyde Ruby. Preliminary design of jml: A behavioral interface specification language for java. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–38, 2006.

[20] Julian Tschannen, Carlo A. Furia, Martin Nordio, and Nadia Polikarpova. AutoProof: Auto-Active Functional Verification of Object-Oriented Programs. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 9035 in Lecture Notes in Computer Science, pages 566–580. Springer Berlin Heidelberg, April 2015. DOI: 10.1007/978-3-662-46681-0_53.

[21] Emmanuel Letier. *Reasoning about agents in goal-oriented requirements engineering*. PhD thesis, PhD thesis, Université catholique de Louvain, 2001.

[22] Carlos Solis and Xiaofeng Wang. A study of the characteristics of behaviour driven development. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 383–387. IEEE, 2011.

[23] Matt Wynne, Aslak Hellesoy, and Steve Tooke. *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2017.

[24] Axel van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pages 249–262, 2001.

[25] Frédéric Boniol and Virginie Wiels. The Landing Gear System Case Study. In Frédéric Boniol, Virginie Wiels, Yamine Ait Ameur, and Klaus-Dieter Schewe, editors, *ABZ 2014: The Landing Gear Case Study*, number 433 in Communications in Computer and Information Science, pages 1–18. Springer International Publishing, June 2014. DOI: 10.1007/978-3-319-07512-9_1.