Messir: A Text-First DSL-Based Approach for UML Requirements Engineering (Tool Demo)

Benoît Ries
University of Luxembourg
Esch-sur-Alzette, Luxembourg
benoit.ries@uni.lu

Alfredo Capozucca
University of Luxembourg
Esch-sur-Alzette, Luxembourg
alfredo.capozucca@uni.lu

Nicolas Guelfi
University of Luxembourg
Esch-sur-Alzette, Luxembourg
nicolas.guelfi@uni.lu

Abstract

This tool paper presents the design and tool-support of Messir, an approach centered on textual domain-specific languages supported by our open-source UML requirements engineering tool, named Excalibur. The novelty of our approach is the actual integration in a single workbench (Excalibur) of textual DSLs richly covering the requirements and analysis phases, i.e. improved use-cases, environment, conceptual and operations models; *and* the read-only visualisation of the requirements with UML-compliant views; *and* the generation of scientific requirements analysis documents in LaTeX; *and* the formal simulation of test cases requirements.

We designed our Messir language, with a grammar-based approach generating a textual editor, using the XText framework as an Eclipse plugin. Messir DSL's static semantics is defined as a set of validation rules guiding end-users through the requirements analysis phase. Messir DSL's semantics is given as a semi-automatic translation to prolog code. We also generate, from the requirements model elements, read-only graphical views (using the Sirius eclipse plugin) as well as a complete requirements analysis document in LaTeX.

This approach and tool have been used as a requirements engineering educational tool in several bachelor and master semesters.

CCS Concepts • **Software and its engineering** → **Domain specific languages**; **Requirements analysis**; **Documentation**; *Formal software verification*;

Keywords Integrated Workbench, Generative Approaches, Model-Driven Engineering, Domain-Specific Languages, Requirements Engineering, LaTeX, Prolog

SLE '18, November 5–6, 2018, Boston, MA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6029-6/18/11…$15.00
https://doi.org/10.1145/3276604.3276614

ACM Reference Format:
Benoît Ries, Alfredo Capozucca, and Nicolas Guelfi. 2018. Messir: A Text-First DSL-Based Approach for UML Requirements Engineering (Tool Demo). In *Proceedings of the 11th ACM SIGPLAN International Conference on Software Language Engineering (SLE '18), November 5–6, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3276604.3276614

1 Introduction

Due to the need for an integrated requirements engineering tool *centered on a textual specification language* targeted to our students at University of Luxembourg, and a tool support for our Messir methodology [5], we have started to develop the Excalibur workbench (and Messir DSLs) in 2012.

One of the main requirements for Excalibur and its DSL was to design a custom textual requirements language having a graphical notation. After having surveyed meta-tools for textual and graphical DSLs modeling, we have chosen the Eclipse framework [1] because of its extensibility, maturity and it is open-source. Moreover, a number of DSL workbenches are available on Eclipse, in particular we have selected XText [4] for the design of our textual DSL, and respectively Sirius [15] for the design of our graphical DSL.

This tool paper begins by a short presentation of the Messir approach's concepts and processes. Then, Section 3 presents the Excalibur workbench architecture. In Section 4, we sketch the design of our Messir DSL allowing to specify our approach's concepts. In Section 5, we describe three generative techniques for our DSL. Lastly, we present related works and a short feedback from students, then we conclude.

2 The Messir Approach

The Messir DSLs presented in this tool paper, are parts of the Messir scientific approach to requirements engineering [5]. Its main contributions are that it is an integrated method supported by a tool (Excalibur), with a flexible requirements specification language, a declarative executable operation language, and improved use-case modeling approach. In short, Messir follows an iterative process composed of a requirements & analysis phase, a documentation phase, and a simulation phase.

The process for the requirements & analysis phase is composed of: firstly, the delimitation of the system and its environment by specifying the actor types, their cardinalities,

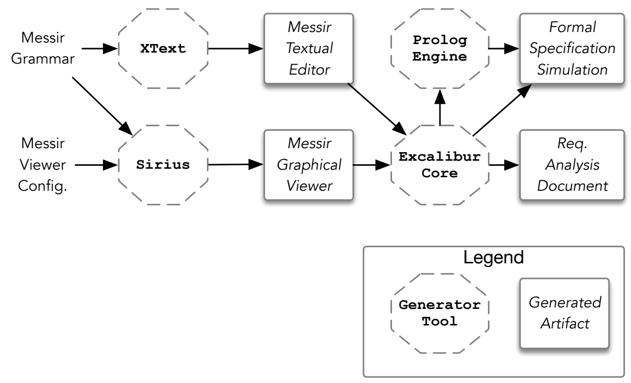


Figure 1. Excalibur workbench architecture

their interfaces with the systems in terms of input/output operations; secondly, the specification of use-cases at different abstraction levels (summary-level, user-goal and subfunction) and the illustration of some system executions by specifying use-case instances; thirdly, the modeling of the system concepts and operations; finally, the modeling of test cases, and some instances to illustrate important test scenarios.

The documentation phase is composed of three steps: the specification of documentation within the requirements specification; the automated generation of a partial requirement analysis document; the manual completion of the general sections of the document, i.e. introduction, conclusion,...

The simulation phase is composed of four steps: the declarative specification of the system operation pre/post conditions with Messir OCL-like DSL and the specification of the test cases; the automated generation of partial prolog code from these specifications; the manual implementation in prolog code of the pre/post conditions based on their specifications; the actual simulation of the test cases, formally evaluating the test steps‘ oracles.

3 Excalibur Workbench Architecture

Excalibur [2] has been developed by the authors as an extension to Eclipse combining the 3 tools, as shown in Fig. 1:

XText [4] which converts an EBNF-like grammar into a full-fledge textual editor, including syntax highlighting, autocompletion and validation rules. This editor has been configured to fit the needs of our DSL, with custom validation rules and auto-completion features, see Section 4.2.

Sirius [15] which displays the textual files written with our DSL in UML-like notations, see Section 5.1.

Excalibur Core which is our own development, implemented in Java and XTend, providing: a dedicated *Outline* allowing to navigate through the specification elements in a tree-view style; a *Requirements Analysis Document* generator, see Section 5.2; a *Formal Specification Simulator*, see Section 5.3.

4 Messir DSL Design

4.1 Grammar

The grammar of the Excalibur DSLs is designed in three parts: firstly, the rules related to the general constructs of the language (Messir DSL); secondly, the constraints rules related to the Messir Constraint Language DSL; thirdly, our documentation language (MessirDoc DSL).

Modularity is available in the Messir DSL with the help of *packages*. Each package is defined in its own textual file (with custom extension .msr) and may contain the specification of one or more models. The underlying models of the Messir method requirements concepts presented in Section 2 are all supported in our DSL. As a first rule for designing the Messir DSL, we have chosen to use keywords in natural languages, for example '`Use Case Model {`' to simply state the beginning of the specification of a use-case model. Another design rule is to provide flexibility in the specification, i.e. the DSL should allow for partial specifications, thus we have designed this rule by setting most grammar rules as being *optional* with the cardinality of (*).

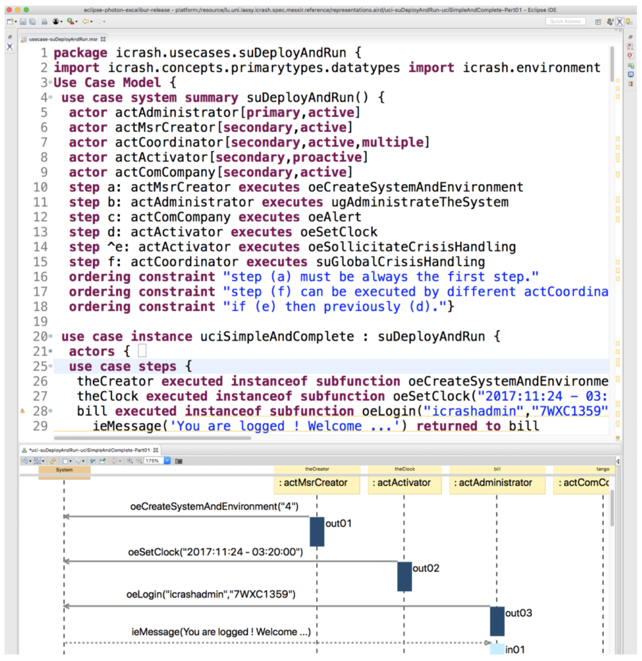
Fig. 2 is an Excalibur screenshot illustrating, with the iCrash [6] case study, the textual specification of a use-case and of an instance of it, together with a generated from the instance specifications. Due to the reduced space of this tool demo paper, it is hardly possible to illustrate all other Messir textual specification models (concept, environment, operation, test), nor their related views.

The Messir Constraint Language (MCL) allows specifying declaratively the operations of Messir elements, its syntax is inspired from OCL [10], and its semantics is defined as a manual translation to prolog. The OCL grammatical concepts offered in MCL are: the navigation through attributes and association ends using dots (.); `if-then-else` conditional expressions; `let-in` declaration expressions; the sending of a message from the system to input operations of actors using (^); collection types and some of their associated operations, e.g. `includes(o)`, `size()`.

The documentation language is a complementary textual language that allows to give natural language descriptions for each of the specification elements written in Messir DSL, e.g. actors, classes, datatypes, ...and also document the created views on these specification elements, e.g. concept model views, use-case views, ...During the report generation process, described in Section 5.2, the content of these descriptions will be taken as input to generate the content of the requirements analysis document.

4.2 Validation Rules

The first set of validation rules are *syntactical validation rules*, these rules are mainly error rules, and inform the specifier when writing specification elements not compliant with Messir. These rules are automatically generated by the XText framework based on the Messir EBNF-like grammar.



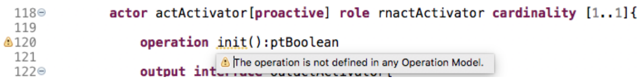
```
package icrash.usecases.suDeployAndRun {
import icrash.concepts.primarytypes.datatypes import icrash.environment
Use Case Model {
 use case system summary suDeployAndRun() {
  actor actAdministrator[primary,active]
  actor actMsrCreator[secondary,active]
  actor actCoordinator[secondary,active,multiple]
  actor actActivator[secondary,proactive]
  actor actComCompany[secondary,active]
  step a: actMsrCreator executes oeCreateSystemAndEnvironment
  step b: actAdministrator executes ugAdministrateTheSystem
  step c: actComCompany executes oeAlert
  step d: actActivator executes oeSetClock
  step ^e: actActivator executes oeSollicitateCrisisHandling
  step f: actCoordinator executes suGlobalCrisisHandling
  ordering constraint "step (a) must be always the first step."
  ordering constraint "step (f) can be executed by different actCoordina
  ordering constraint "if (e) then previously (d)."}

 use case instance uciSimpleAndComplete : suDeployAndRun {
  actors {
  use case steps {
   theCreator executed instanceof subfunction oeCreateSystemAndEnvironme
   theClock executed instanceof subfunction oeSetClock("2017:11:24 - 03:
   bill executed instanceof subfunction oeLogin("icrashadmin","7WXC1359"
     ieMessage('You are logged ! Welcome ...') returned to bill
```

eclipse-photon-excalibur-release - platform:/resource/lu.uni.lassy.icrash.spec.messir.reference/representations.aird/uci-suDeployAndRun-uciSimpleAndComplete-Part01 - Eclipse IDE
usecase-suDeployAndRun.msr
*uci-suDeployAndRun-uciSimpleAndComplete-Part01
175%
System
theCreator
: actMsrCreator
theClock
: actActivator
bill
: actAdministrator
tango
: actComCo
oeCreateSystemAndEnvironment("4")
out01
oeSetClock("2017:11:24 - 03:20:00")
out02
oeLogin("icrashadmin","7WXC1359")
out03
ieMessage(You are logged ! Welcome ...)
in01

Figure 2. An Excalibur screenshot with use-case textual specifications and a use-case instance graphical view

We have defined 50 runtime validation error/warning rules that we use as educational means to inform and teach the specifiers on how to better follow our methodology :

- *warning rules* are mainly used in Excalibur to let the end-user know about future steps to be done, or particular aspects of the methodology not to be overlooked. An example warning validation rule is:
  - *“The operation is not defined in any Operation Model”*, see Figure 3, this warning appears when the operation in question has not (yet) been specified by the end-user. Thus, when the analyst declares an operation, let’s say in an actor, the declared operation will automatically be tagged with this warning to inform that this operation is declared, but not yet specified.



```
       actor actActivator[proactive] role rnactActivator cardinality [1..1]{

           operation init():ptBoolean

           output inte
```

The operation is not defined in any Operation Model.

Figure 3. Warning validation rule in action

- *error rules* are mainly used in Excalibur to block the specifier in his/her requirements engineering process,
  - An example is: *"Infinite loop: use case can’t execute itself !"*. As use-cases may refer to other use-cases of the same abstraction level, it may happen that infinite loop are inadvertently specified.

5 Generative Techniques for a Textual Requirement Engineering Tool

Excalibur follows a generative approach. Requirements analysts can take advantage from three types of generative techniques. Firstly, viewing textual specifications with the help of a graphical DSL. Secondly, generating partial *and extensible* documentation for customers, e.g. for contractual purposes. Thirdly, giving a semantics to the requirements specifications in a formal language.

5.1 Views

In our textually-centered approach, the textual requirements specification provide the complete specifiation. The viewing features are offered to allow *graphical illustrations of certain important aspects* of the models.

The analyst may place the specified concepts that he/she wishes to illustrate on the views and decides on the layout design of the selected elements. Note that we intentionally do not allow specifiers to modify the specification elements from the graphical views, such that all modifications are made in the textual files and that our text-first approach is ensured.

In terms of UML notation, our views are using the syntax of use-case diagrams, sequence diagrams and class diagrams. In our approach, these diagrams are used with the following purposes :

- *use-case views* illustrate the specified actors and their relations to the system operations.
- *use-case instance views* illustrate some carefully selected scenarios of the system executions.
- *concept model views* illustrate the system’s structural concepts: class types, datatypes,. . .
- *environment model views* illustrate how actors interact with the system by sending and receiving events.
- *operation scope views* illustrate for a given system operation, all the types related to it.

5.2 Requirements Analysis Document

The generation of requirements analysis specification documents takes as input the specified elements with the Messir DSLs as well as the views created by the specifier to generate a LaTeX document. The generation implementation has been writen in XTend, an object-oriented general-purpose language well-suited to implement model-to-text transformations. We have designed the transformation to give the flexibility to enhance the generated LaTeX document by isolating the generated LaTeX code from the manually added LaTeX parts. Excalibur offers a flexible report generation process adapted to different categories of readers:

- *definition-level*: the generated report includes all documented specification elements described in natural language; it also includes all documented views.

- *specification-level* the generated report includes all content from the definition-level, and the specification of operations and types in MCL.
- *simulation-level* the generated report includes all content from the specification-level, and the prolog code describing formally the semantics of the operations and types.

5.3 Formal Specification Simulation

The Messir DSL allows to write test cases and instances of test cases. The formal simulation specification generator creates a prolog simulation project from the requirements specified in Messir DSL, containing:

- *MESSAM prolog code*: the Messir Abstract Machine (MESSAM), which is our prolog implementation of the Messir DSL metamodel, that must be part of all Excalibur simulation projects.
- *Types specification*: all specified types and actors in prolog in a way that is compliant with MESSAM.

Some parts of the prolog project must be completed manually after the simulation project generation :

- the *operation pre/post conditions* must be implemented in prolog manually based on the MCL operation specification from the requirements project.
- *test cases specification*: must also be translated manually from the test case models specified in Messir textual requirements.

6 Related Work

We have not found any related work on integrated workbenches providing altogether DSLs covering a rich coverage of the requirements and analysis phases, i.e. improved use-cases, environment, conceptual and operations models; *and* report generation in LaTeX; *and* formal test cases requirements simulation in Eclipse, as in our Messir approach.

The ReSA tool [9] provides an integrated environment for specification of structured requirements with SAT-based verification. This work targets EAST-ADL models (i.e. not UML). Moreover, no report can be generated from the requirements.

Hoffman et al. defined Nautilus [8], a textual requirements DSL structurally compliant with UML, supported by the ViPER tool. The DSL is solely focused on the specification of use-cases, in particular, Nautilus DSL is not covering the specification of concepts, system operations, nor system actors. There is no support for simulation, nor report generation.

The work by Savic et al. on the SilabReq DSL [13] is similar to our work, in the sense that it defines a structured DSL for the specification of use-cases and the related concepts, system operations, and actors and allows interpretation and execution of the specified requirements. The main differences are: firstly, the system operations in SilabReq are specified with an imperative language unlike Messir which is providing an OCL-like declarative language; secondly, the Messir DSL offers a wider requirements specification coverage by providing ways to specify use-case instances, test cases and test cases instances and their related views with a syntax inspired from UML sequence diagrams; thirdly, no report generation is offered; lastly, a tool-support prototype is described in [3], unfortunately with neither graphical support, nor report generation.

Since the recent formalization of fUML [12] and its reference implementation, named Alf [11] provided by the OMG, some UML tools have offered some textual support, compliant with the fUML standard, as for instance MagicDraw [14] and Paypyrus with the Moka framework [7]. These works differ from ours by remaining graphically-centered approaches.

7 Students Feedback

Since its first release to students in 2013, Excalibur and its accompanying Messir DSLs have been used in more than 10 semesters for bachelor and master programs at University of Luxembourg and partner institutions: Univ. of Rosario, Innopolis Univ. and St Petersburg Polytechnic Univ.

Our students surveys on the lectures using Messir/Excalibur resulted, out of 90+ students answers, in a majority of the students agreeing (or strongly agreeing) both on recommending the lectures to others, and on the statement that the learning resources met their needs. Shortly, what students liked most were: the hands-on approach working on a project with tool-support, and some particularly liked the Excalibur automatic report generation. The students with negative comments on the tools were mostly about the presence of bugs in the tool. . . an unfortunate charactestic of software applications.

8 Conclusion

In this tool demo paper, we presented our solution for a requirements engineering tool, named Excalibur, supporting our methodology, that is centered on the Messir *textual* DSL having typical features of textual editors (thanks to XText) for which we have developped 50 custom validation rules to guide the analyst during the requirements elicitation phase.

Excalibur implements three generative techniques to make the best use of the textual requirements specifications, firstly by generating read-only views in a UML-style (thanks to Sirius), secondly by generating an extensible requirements analysis document compiling all textual and graphical requirements information; lastly by generating a partial prolog implementation supporting the DSL metamodel for simulation purposes.

Acknowledgments

The authors would like to thank all the students from University of Luxembourg and from our partner institutions for having provided valuable feedback during our lectures, as well as bachelor and master theses, for which the Excalibur tool and the Messir DSL were used.

References

[1] 2018. Eclipse foundation website. Retrieved September 27, 2018 from http://www.eclipse.org.

[2] 2018. Messir and Excalibur website. Retrieved September 27, 2018 from https://messir.uni.lu.

[3] Alberto Rodrigues da Silva, Sinia Vlajic, Saa Lazarevic, Ilija Antovic, Vojislav Stanojevic, and Milo Milic. 2014. Preliminary Experience Using JetBrains MPS to Implement a Requirements Specification Language. In *2014 9th International Conference on the Quality of Information and Communications Technology*. IEEE, Guimaraes, Portugal, 134–137.

[4] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: Implement Your Language Faster Than the Quick and Dirty Way. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA '10)*. ACM, Reno/Tahoe, Nevada, USA, 307–309.

[5] Nicolas Guelfi. 2016. *The Messir Scientific Approach to Requirements Engineering*. Laboratory for Advanced Software Systems Technical Report TR–LASSY–16–01. University of Luxembourg. Retrieved September 27, 2018 from http://messir.uni.lu.

[6] Nicolas Guelfi. 2018. iCrash: A Crisis Management Case Study - Messir Analysis Document. Retrieved September 27, 2018 from https://messir.uni.lu/confluence/display/EXCALIBUR/Downloads.

[7] Sahar Guermazi, Jérémie Tatibouet, Arnaud Cuccuru, Saadia Dhouib, Sébastien Gérard, and Ed Seidewitz. 2015. Executable Modeling with fUML and Alf in Papyrus: Tooling and Experiments. Ottawa, Canada.

[8] Veit Hoffmann, Horst Lichter, Alexander Nyßen, and Andreas Walter. 2009. Towards the Integration of UML- and Textual Use Case Modeling. *The Journal of Object Technology* 8, 3 (2009), 85.

[9] Nesredin Mahmud, Cristina Seceleanu, and Oscar Ljungkrantz. 2016. ReSA Tool: Structured Requirements Specification and SAT-Based Consistency-Checking. 1737–1746.

[10] OMG. 2006. *Object Constraint Language (OCL)*. Full Specification formal/06-05-01. Object Management Group.

[11] OMG. 2017. *Action Language for Foundational UML (Alf) Concrete Syntax for a UML Action Language Version 1.1*. Full Specification formal/2017-07-04. Object Management Group.

[12] OMG. 2017. *Semantics of a Foundational Subset for Executable UML Models Specification v1.3*. Full Specification formal/17-07-02. Object Management Group.

[13] D. Savic, S. Vlajić, S. Lazarević, I. Antović, S. Vojislav, M. Milić, and A. Silva. 2015. Use case specification using the SilabReq Domain Specific Language. *Computing and Informatics* 34 (2015), 877–910.

[14] Ed Seidewitz. 2017. A Development Environment for the Alf Language within the MagicDraw UML Tool (Tool Demo). ACM Press, 217–220.

[15] V. Viyović, M. Maksimović, and B. Perisić. 2014. Sirius: A Rapid Development of DSM Graphical Editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*. 233–238.