Messir: A Text-First DSL-Based Approach for UML Requirements Engineering (Tool Demo)
Benoit Ries, Alfredo Capozucca and Nicolas Guelfi
University of Luxembourg, Esch-sur-Alzette, Luxembourg

2. Messir Textual DSLs

Messir Constraint Language
- declarative specification of operations
- syntax inspired from OCL
- semantics defined as a manual translation to prolog
- covered concepts include: navigation, conditional expressions, messages sending

Messir Documentation Language
- complementary textual language
- natural language descriptions used during report generation
- automatic documenting Messir specification elements
- automatic documenting Excalibur views

Validation Rules
- A number of syntactical validation rules are generated automatically by the XText framework based on the Messir DSL grammar.
- We have implemented 50 additional runtime validation rules used as educational mounts.
- warning rules are meant to let the end-user know about future steps to be done, or particular aspects of the methodology to not be overlooked.
- error rules are meant to block the end-user in his requirements specification process.

3. Generative Techniques

View Generation
- read-only views
- illustrate certain aspects of the textual requirements
- supported views are: use-case, use-case instance, concept model, environment model, operation scope, test case, test-case instance
- their concrete syntax is based on UML use-case, class and sequence diagrams
- these views are integrated in the requirements analysis document during the report generation phase.

Report Generation
- Takes as input:
  - requirements elements in Messir
  - documentation in MessirDoc of the elements and views
  - actual views created in the requirements project
- Generates a PDF document to be completed, e.g. with introduction, conclusion, etc.
- Flexible process:
  - definition-level mainly contains natural language descriptions and documented views
  - specification-level additionally includes declarative operations specifications in MCL
  - simulation-level additionally includes prolog code of the operations and types semantics.

Simulation
- Takes as input:
  - text cases and instances of test cases specified in Messir
  - generates a prolog simulation project containing:
  - MESSAM prolog code: the MESSAm Abstract Machine, which is our prolog implementation of the Messir DSL metamodel
  - operation pre/post conditions
  - and the test cases specification

Work Summary

Messir is a scientific approach, yet flexible, for the specification of UML requirements. It is supported by the Excalibur tool, used for software engineering education, since 2012 in numerous institutions for project-based lectures. The Excalibur tool provides as an integrated workbench, the possibility to describe rich textual UML requirements & analysis specifications, to generate a structured report in LATEX, and to formally simulate, with a prolog engine, the test cases specified in the requirements.

References

Students Feedback

Our students surveys on the lectures using Messir/Excalibur resulted, out of 90+ answers, in a majority of students agreeing both on "recommending the lectures to others", and that "the learning resources met their needs". Positive comments were "the integrated hands-on approach" and "the report generation". Negative comments were mostly about "the actual presence of bugs in the tool".

Conclusion

This poster presents our solution for a requirements engineering tool, named Excalibur, supported by the Messir MetaModel (that is centered on the Messir textual DSLs having typical features of textual editors (thanks to XText) for which we have developed 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 executable requirements analysis document compiling all textual and graphical requirements information; lastly by generating a partial prolog implementation supporting the DSL metamodel for simulation purpose.

Acknowledgements

The authors would like to thank all the students from University of Luxembourg (Luxembourg), University of Rosario (Argentina), Imolaer University (Russia) and St. Petersburg Polytechnic University (Russia) for their help and support in the usage of the tool and bug reporting that greatly helped making more stable releases.