reqT.org – Towards a Semi-Formal, Open and Scalable Requirements Modeling Tool

Björn Regnell

Dept. of Computer Science, Lund University, Sweden
bjorn.regnell@cs.lth.se

Abstract. [**Context and motivation**] This research preview presents ongoing work on a free software requirements modeling tool called reqT that is developed in an educational context. [**Question/problem**] The work aims to engage computer science students in Requirements Engineering (RE) through a tool that captures essential RE concepts in executable code. [**Principal ideas**] Requirements are modeled using an internal DSL in the Scala programming language that blends natural language strings with a graph-oriented formalism. [**Contribution**] The metamodel of reqT and its main features are presented and modeling examples are provided together with a discussion on initial experiences from student projects, limitations and directions of further research.

Keywords: requirements engineering, requirements modeling, software engineering, CASE tool, requirements metamodel, requirements engineering education, internal DSL, embedded DSL, Scala programming language.

1 Introduction

There are many challenges in teaching Requirements Engineering (RE) [6, 9], including conveying requirements modeling skills [1]. Given a wide-spread attention on agile methods with less emphasis on extra-code artifacts [8], it may be particularly challenging to motivate coding-focused engineering students (and software practitioners) to spend serious effort on requirements modeling. One way to inspire software engineers to learn more about and do more RE may be to offer an interesting software tool. There are nowadays numerous commercial RE tools available, but many are expensive, complex and not sufficiently open [2].

This paper presents on-going work on a tool named reqT that aims to provide a small but scalable, semi-formal and free software package for an educational setting that (hopefully) can inspire code lovers to learn more about requirements modeling. A long-term goal of reqT is to offer an open platform for RE research prototypes, e.g. for feature modeling and release planning research. The tool development started in 2011 at Lund University, where reqT is used in RE teaching at MSc level in the Computer Science & Engineering program. In 2012 reqT was rebuilt from scratch based on student feedback. The tool can be downloaded from: `http://reqT.org`

The paper is organized as follows. Section 2 states the objectives and motivates the design strategy of reqT. Section 3 presents the metamodel of reqT and some example reqT models. Section 4 discusses limitations and some initial experiences from using reqT in teaching and concludes the paper with a sketch of future research directions.

J. Doerr and A.L. Opdahl (Eds.): REFSQ 2013, LNCS 7830, pp. 112–118, 2013.

2 Goals, Design Strategy and Rationale

The main objective behind reqT is to establish a set of essential RE concepts and capture them in an expressive, extensible and executable language appealing to computer science students (and eventually practitioners). This general objective is accompanied by the following main goals and associated design strategies:

1. **Semi-formal.** *Goal:* Provide a semi-formal representation of typical requirements modeling constructs that can illustrate a flexible combination of expressive natural language-style requirements with type-safe formalisms allowing static checks. *Design:* Use graph structures based on typed nodes representing typical requirement entities and attributes, and typed edges representing typical requirements relations, and implement the graph as an associative array (map). *Why?* Graphs are well-known to many CS students. Maps are efficient from an implementation perspective and may be less complex to master compared to e.g. SQL databases.
2. **Open.** *Goal:* Provide a platform-independent requirements tool that is free of charge. *Design:* Use Java Virtual Machine technology and release the code under an open source license. Use tab-separated, tabular text-files for import and export. Use HTML for document generation. *Why?* There are many free libraries available that runs on a JVM. Tab-sep and HTML support interoperability.
3. **Scalable.** *Goal:* Provide an extensible requirements modeling language that can scale from small, concise models to large families of models with thousands of requirements entities and relations. *Design:* Implement reqT as an internal DSL (Domain-Specific Language) in the Scala programming language [7]. Use Map and Set from Scala collections to represent requirements graphs. *Why?* Scala is a modern, statically typed language with an efficient collections library. Scala offers scripting abilities that provide general extensibility without re-compilation. Integrated development environments [11], as well as interactive scripting tools are available [3].

These goals, design strategies and rationale are directing the on-going work, and it remains to be investigated to what extent the main objective and goals can be met. A critical issue is how to interpret what are ”essential” RE concepts and ”typical” modeling constructs. The reqT tool is used in a course based on a specific text book [4] and a specific student project concept [5], and the concepts of the reqT requirements metamodel (see Fig. 2) reflect that context. However, the reqT architecture is prepared for extensions of new concepts in the metamodel to cater for different educational contexts.

3 Modeling Requirements with reqT

A reqT model includes sequences of graph parts `<Entity><Edge><NodeSet>` separated by comma and wrapped inside a `Model( )` construct. A small reqT Model with three `Feature` entities and one `Stakeholder` entity is shown below:

```
Model(
  Feature("f1") has (Spec("A good spec."), Status(SPECIFIED)),
  Feature("f1") requires (Feature("f2"), Feature("f3")),
  Stakeholder("s1") assigns(Prio(1)) to Feature("f2")
)
```

The corresponding graph implied by the above model is depicted in Fig. 1. The edges represent different relations between entities, in this case the `requires` and `assigns` relations. Nodes with outgoing edges are called *sources* and nodes with incoming edges are called *destinations*. There is a special edge called `has` that is used to attach attributes to entities. The different types of entities, relations and attributes of the reqT metamodel, depicted in Fig. 2 can be combined freely, although a has-edge can only link to attributes, while a relation can only link to entities. In the metamodel of reqT in Fig. 2, abstract types are shown in italics and concrete types are shown in bold. All concrete types are Scala case classes [7]. All entities have a string-valued id for unique identification. Most attributes have string values that can be used to combine informal, natural-language expressions with formal graph-structures. The `Status` attribute can be associated to entities to reflect their degree of refinement in RE and down-stream processes by different `Level` values, as depicted in Fig. 3.

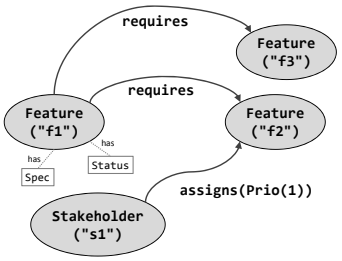


Fig. 1. A reqT model depicted as a graph

Domain-level task descriptions [4] can be modeled using the scenario-type requirement entity `Task`, as shown in the model below. This example[1] is modified from Lauesen [4], p. 93. The special relation `owns` is used to express hierarchical decomposition and reqT ensures that an entity only can be owned by one other entity.

```
var m = Model(
  Task("reception work") owns (Task("check in"), Task("booking")),
  Task("check in") has (
    Why("Give guest a room. Mark it as occupied. Start account."),
    Trigger("A guest arrives"),
    Frequency("Average 0.5 check-ins/room/day"),
    Critical("Group tour with 50 guests.")
  ),
  Task("check in") owns (
    Task("find room"), Task("record guest"), Task("deliver key")),
  Task("record guest") has Spec(
    "variants: a) Guest has booked in advance, b) No suitable room"
  )
)
```

There are a number of operators defined for reqT models including the aggregate, restrict and exclude operators denoted by double plus `++` and slash `/` and backslash `\` respectively. The expression `m1 ++ m2` results in a new aggregated model with `m1` and `m2` merged, with parts of `m2` potentially overwriting overlapping parts of `m1`. The restrict and exclude operators produce new submodels including or excluding parts of a `Model` based on the right operand argument that can be of different `Element` types, as explained subsequently.

[1] For more examples on how to combine various entities and relations of reqT into different requirements modeling styles, see: `http://reqT.org/`

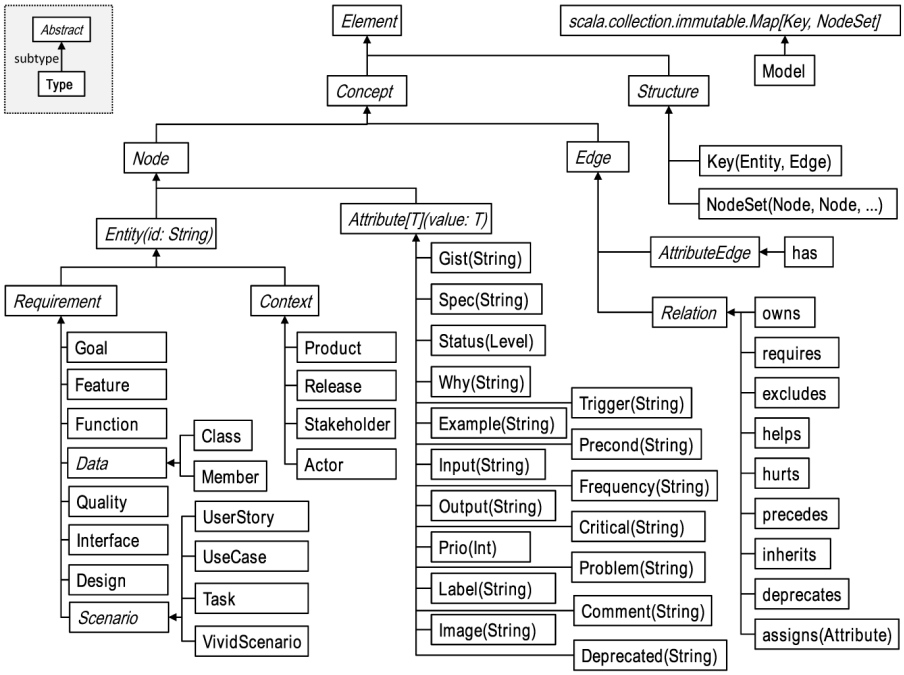


Fig. 2. The reqT version 2.2 metamodel

The expression `m / Task("x")` results in a new submodel that is restricted to all parts of `m` with `Task("x")` as source node, while the expression `m \ Task` results in a new submodel that excludes all `Task` sources. The expression `((m / e) ++ (m \ e) == m)` is always true.

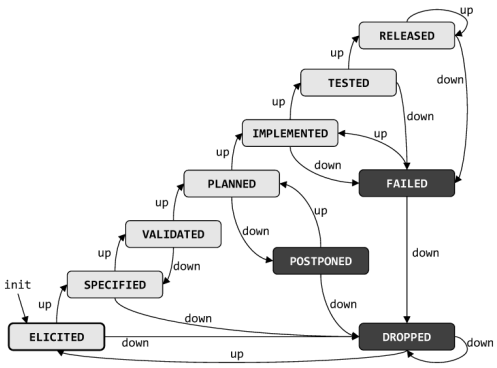


Fig. 3. Refinement levels of the Status attribute

A reqT `Model` has the methods `up` and `down` that promote or regress all its `Status` attributes according to the state machine in Fig. 3. By using `/` and `\` for extracting submodels, levels can be selectively promoted, e.g. the expression `m = (m / Feature("x")).up ++ (m \ Feature("x"))` updates `m` to a new model where only `Feature("x")` is promoted to the next level. Several more operators and methods for create/read/update/delete of entities using Scala scripts are available in the `Model` case class, see further: `http://reqT.org/`

In our course projects [5] students shall produce requirements documents that can be validated by laymen. This is supported in reqT by an export operation on models called `toHtml` that generates files that can be shown in web browsers as illustrated in Fig. 4. The HTML generation is controlled by a DocumentTemplate case class that allows for specifying title, free form text paragraphs and optional numbers of Chapters containing optional Sections including specified parts of a reqT model in flexible ways using Scala function literals that can, e.g., apply restrict and exclude operators to models. In Fig. 4 the Scala function literal `m => m / Context` restricts the contents of a chapter to only `Context` type source entities. Fig 4 also shows `toTable` export to spreadsheet programs via tab-separated text files. The code in Fig. 4 can be executed e.g. as a script using the interactive Scala Read-Evaluate-Print-Loop (REPL) from the command line, or in a scripting environment such as Kojo [3], or inside the Scala Eclipse IDE [11].

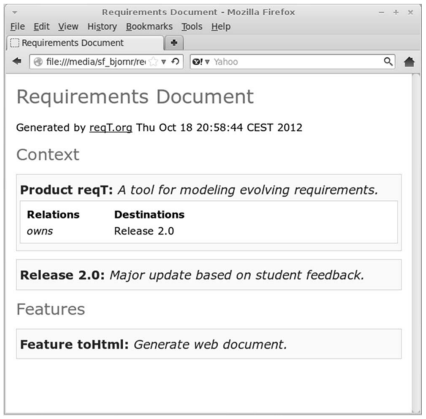
```
var m = Model(
  Product("reqT") has
    Gist("A tool for modeling evolving requirements."),
  Release("2.0") has
    Gist("Major update based on student feedback."),
  Product("reqT") owns Release("2.0")
)

m += Feature("toHtml") has Gist("Generate web document.")

val myTemplate = DocumentTemplate(
  "Requirements Document",
  Text("Generated by " +
    " <a href=\"http://reqT.org\">reqT.org</a> " +
    ( new java.util.Date ) ),
  Chapter("Context", Text(""), m => m / Context),
  Chapter("Features", Text(""), m => m / Feature)
)

m.toHtml(myTemplate).save("reqT.html")

m.toTable.save("reqT.txt")
```



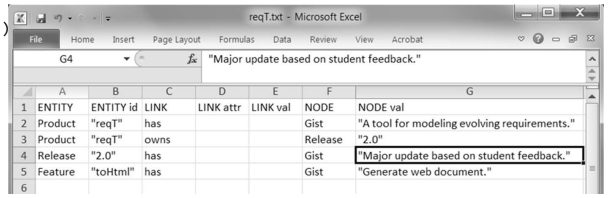


Fig. 4. Example of template-based export to HTML and tab-separated table export

4 Discussion and Conclusion

The results of the on-going work with reqT remains to be further investigated and a validation of reqT as a RE learning tool and research experimentation platform is subject to future work. This section discusses some preliminary experiences, limitations, relation to state-of-the-art and future research directions.

Preliminary Proof-of-concept. The first version of reqT was tried on a voluntary basis by 12 students working in groups of 6 students each during fall 2011. Statements from course evaluations indicate that the students found reqT useful in their learning. One

group used a configuration management tool for reqT models to manage their parallel work, while one group used a cloud service and tab-sep export/import to collaborate over the Internet. The group with the largest requirements model produced 64 features, 18 tasks, 12 functions, 30 data requirements and 33 quality requirements, in total 157 requirements entities.

Several students appreciated that reqT can mix informal text with a graph-oriented formalism, but some requested more elaborated functionality for document generation, as well as linking to external images. Some students also requested more modeling examples that show how the text book techniques could be transferred to reqT models.

Based on student feedback, reqT was rebuilt from scratch during 2012 with a new architecture and a new version of the meta model (see Fig. 2), as well as a revised Scala-internal DSL. The template-controlled HTML generation was implemented based on student suggestions. The teaching material was complemented with more example models directly related to the textbook. The second version of reqT is currently tested by students in a new course instance and a post-course evaluation of reqT is planned in spring 2013.

Our preliminary experiences from applying reqT in teaching suggest that reqT, if used in a suitable teaching context, may encourage students with a code-focused mind set to learn and practice RE in the following ways: (1) A free and platform-independent software tool that is implemented using a modern programming language with interactive scripting facilities can attracts the interest of code-focused students. (2) Requirements can be processed, queried, transformed or exported using Scala scripts, and the open-ended nature of reqT that allows students to code their own scripts to both manage requirements models and to adapt reqT to fit their RE needs in the course project was appreciated by several coding-literate students. (3) By turning requirements models into executable code, students can use programming tools such as a console command line interpreter (the Scala REPL) as well as a source code version control system (e.g. git-scm.com) to branch and merge their collaborative work on requirements in ways they are used to from their previous collaborative software implementation courses, including issue tracking systems and code review support.

Relation to State-of-the-Art. To the best of our knowledge there is no other RE tool that allows semi-formal requirement models to become executable programs through an internal Scala DSL, and thus letting coding, testing and requirements engineering share the same media. In the RE tool survey by Carrillo de Gea et al. [2] it is pointed out that ”many expensive tools aren’t sufficiently open”. The reqT technology aims to be completely free and open to facilitate academic usage, collaborative evolution and incorporation of new RE concepts in different teaching and research contexts. Many of the existing tools have proprietary representations [2], while users of reqT can extend the reqT metamodel with new entities and attributes simply by adding case classes with a few lines of code. However, reqT cannot compete with versatile commercial RE tools [2] in terms of e.g. features completeness and graphical user interface.

Limitations. In its current version, reqT has a number of limitations: (1) As the user interface is text based and depends on the command line interface of the Scala REPL or a script editor environment [3, 11], students that only are prepared to use graphical

user interfaces may be discouraged. Some of our students preferred to work in a GUI spreadsheet application using tab-separated exports from reqT that was generated by other team members assigned by the student group to be reqT experts. (2) It requires some knowledge of Scala to tailor reqT exports and there is a need for a more comprehensive API for adaptable document generation. (3) The embedded DSL requires some learning efforts and it remains to be investigated if the effort is justified by the knowledge gained. (4) To support scalability to large families of reqT models there is a need for modularization concepts and overview visualizations. (5) The explicit typing of entities with keywords such Feature and Stakeholder can be perceived as verbose compared to more concise but potentially cryptic abbreviations (e.g. Fe, Sh). This may be addressed by DSL-specific editor support, such as code-completion, code folding and code templates.

Future Work. Further directions of research include (1) incorporation of constraints on models for support of prioritization and release planning [10], (2) more elaborate semantic checks to better guide requirements modelers, and (3) graphical visualization of requirements graph models. (4) Natural Language Processing technology including e.g. ambiguity risk detection may be interesting to combine with reqT. (5) It is also important to further investigate the pedagogic advantages and limitations of the approach.

A major objective of this research preview paper is to expose the latest version of reqT to the community of RE scholars and to invite discussions and contributions.

Acknowledgments. This work is partly funded by VINNOVA within the EASE project.

References

1. Callele, D., Makaroff, D.: Teaching requirements engineering to an unsuspecting audience. In: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2006, pp. 433–437 (2006)
2. Carrillo de Gea, J., Nicolas, J., Aleman, J., Toval, A., Ebert, C., Vizcaino, A.: Requirements engineering tools. IEEE Software 28(4), 86–91 (2011)
3. Kogics: Kojo, `http://www.kogics.net/kojo` (visited November 2012)
4. Lauesen, S.: Software Requirements - Styles and Techniques. Addison-Wesley (2002)
5. Lund University: `http://cs.lth.se/ets170` (visited November 2012)
6. Memon, R.N., Ahmad, R., Salim, S.S.: Problems in requirements engineering education: a survey. In: Proceedings of the 8th International Conference on Frontiers of Information Technology, FIT 2010, pp. 5:1–5:6. ACM (2010)
7. Odersky, M.: et al.: An overview of the Scala programming language. Tech. rep (2004), `http://lampwww.epfl.ch/~odersky/papers/ScalaOverview.html`
8. Ramesh, B., Lan, C., Baskerville, R.: Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal 20(5), 449–480 (2010)
9. Regev, G., Gause, D.C., Wegmann, A.: Experiential learning approach for requirements engineering education. Requirements Engineering 14(4), 269–287 (2009)
10. Regnell, B., Kuchcinski, K.: Exploring software product management decision problems with constraint solving - opportunities for prioritization and release planning. In: 2011 Fifth International Workshop on Software Product Management, IWSPM, pp. 47–56 (2011)
11. Scala Eclipse IDE: `http://scala-ide.org/` (visited November 2012)