Using Difference Information to Reuse Software Cases

Jürgen Ebert, Daniel Bildhauer, Hannes Schwarz, Volker Riediger
Institut für Softwaretechnik, Universität Koblenz-Landau
(ebert|dbildh|hschwarz|riediger)@uni-koblenz.de

1 Introduction

The goal of the *ReDSeeDS*[1][2] (Requirements-Driven Software Development System) project is to support reuse of software development artifacts and code in a model driven development context.

The set of all *artifacts* that are produced during the development of a software product together with all their *interconnections* is called a *software case* in the context of ReDSeeDS. A software case contains (at least) a set of requirements, an architectural model, several detailed design models, the produced code and some transformational information – the latter describing potential model-to-model and/or model-to-code transformations. Based on similarity of new requirements to those stored in former software cases partial solutions shall become reusable. Figure 1 (taken from the ReDSeeDS website) sketches this approach.

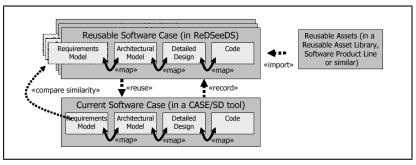

Figure 1. ReDSeeDS reuse of software cases

In this note, we sketch how versioning for artifacts is combined with a difference-based approach for software case retrieval.

2 Application Scenario

The artifacts constituting a software case are written using different languages. The requirements shall be formulated using the *ReDSeeDS Requirements Specification Language* (RSL, a first proposal of which has just been defined), the models are written using some profiled version of *UML* and the code will be in some *object-oriented programming language*. The languages used for the software case artifacts are dependent on the development technology and the development environment used. To make the ReDSeeDS approach widely usable, a wide range of languages for writing/generating artifacts will have to be supported. This is done by using a homogeneous methodology for developing *metamodels*. A UML-MOF-Style is followed. Thus, all languages defined in UML 2.0 can also be used in a software case (almost) without change. But other *non*-UML-languages will be explicitly includable, as well.

All artifacts of software cases and their interconnections are kept in a common environment which stores the artifacts themselves in some versioned *artifact repository* and their structure and interconnections are placed into an additional *fact repository*. All artifacts have unique identifiers and are versioned. The *artifact repository* has to keep track of the artifacts that constitute a software case and their version and configuration information. Besides this, the abstract syntax graphs of the artifacts and their interconnection will be stored explicitly in the *fact repository*, which will be kept synchronous with the artifact repository (Figure 2) by an appropriate extraction mechanism.

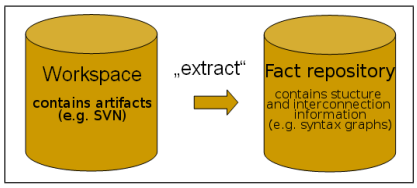

Figure 2. Relation of artifact and fact repository

Since a model-driven approach to software development is being pursued, the mappings/transformations used supply much more traceability/interconnection information than conventional software development does. All this intermediate information accumulated during the transformation-driven software development process is included in the fact repository, as well.

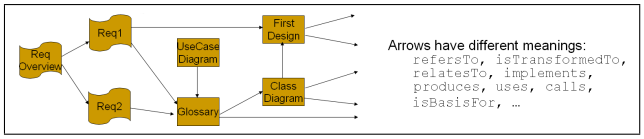

Figure 3. Interconnections in fact repository

The fact repository can be seen as one large graph, called the *software case graph (SC-graph)*, which is assumed to be the abstract view of all documents in the artifact repository and their interconnectedness (Figure 3). All artifacts

[1]This work was supported in part by Commission of the European Communities in the STREP-project *ReDSeeDS* contract 33596.
[2]http://www.redseeds.eu/

are represented by some (sub)graph in the SC graph. The graph-based library JGraLab is used to implement the fact repository. JGraLab supports TGraphs, i.e. typed, directed, and attributed ordered graphs. TGraphs are amenable to algorithmic problem solution and can be easily accessed by queries.

The ReDSeeDS metamodel describes the abstract syntax of the artifacts and thus the class of *abstract syntax graphs* that represent these artifacts in the repository. Furthermore, it describes the possible interconnections between these graphs. It thus acts as the *schema* of the fact repository.

3 Similarity, Versioning, and Differencing

Requirements descriptions usually consist to a large part of natural language sentences. Similarity of software requirements can be based on their textual contents, their abstract structure or both. While *information retrieval research* gives approaches to text similarity, the abstract structure similarity approach based on syntax graphs should profit from research work on *graph similarity research*.

In the ReDSeeDS context artifacts are edited by several different tools, e.g. RSL-texts are edited by some RSL-tool, UML-models are edited by some UML-tool (in fact, Enterprise Architect[3] is used at present), and program code will be partly generated and partly edited by some IDE. All these artifacts will be subject to version management on the granularity of complete artifacts in the *artifact repository*.

In contrast to this, the *fact repository* always reflects *one* configuration of the artifacts, i.e. one baseline. Whenever a new artifact version shall be included into it, this can be done by partial replacement of this artifact's graph in the repository. Mirroring abstract representations of artifacts in a fact repository is a state of the art technique in reengineering environments. Partial replacements of artifact graphs can be implemented using a query-based approach, as long as appropriate metamodel-information is available.

The vision of ReDSeeDS is the reuse of software cases based on their requirement description. The computing of artifact *differences* will be executed on the fact repository, i.e. on the basis of interconnected syntax graphs. Thus, graph theoretic approaches for computing differences will be used.

Given a set of stored software cases there are two possible reuse scenarios, which might be combined in practice. First, a new project can be started on the basis of some stored case, taking its RSL-description R, and modifying it to a variant R'.

[3] http://www.sparxsystems.com/products/ea.html

In this case, the differences between R and R' may be extracted from the (log of the) editing procedure. Second, a new project might be started totally from scratch by writing a new set of RSL-specifications.
In this case, the new specification R' has to be compared to all existing specifications in order to find some (most similar) specification $R*$. Then, the differences between R' and $R*$ have to be computed afterwards by some difference-algorithm.

Given two RSL-documents R and R', a differencing algorithm should be able to compute at least the common part $\overline{R} = R \sqcap R'$ of both documents, which is assumed to be a vertex-set generated subgraph. Since the overall SC-graph contains $\overline{R}$ as a subgraph, it is assumed that it is the "seed" of the common parts of the corresponding software cases. All vertices reachable from vertices of $\overline{R}$ by appropriate traceability links are called $\mathbf{slice}(\overline{R})$, the *slice* of the SC-graph determined by R. The computation of the slice depends on the definition of traceability. The computation can be done by an appropriate query.

Since it cannot be assumed that the subgraph defined by the slice is already a meaningful set of documents, a superset, called *closure* $\mathbf{closure}(\mathbf{slice}(\overline{R}))$ has to be computed then, which fulfills the necessary consistency conditions of a software case and can be shown to the ReDSeeDS user as a set of reusable artifacts.

4 Conclusion

This note described the application of model differences and model similarity for reusing software in the ReDSeeDS project. While the main concepts and the overall approach have already been defined, there are still some open questions which will have to be answered during the project. The first one covers the adoption or development of an algorithm to find the most similar software case R* for a given case R solely based on their requirement sets. Another issue is the combination or extension of existing algorithms to compute the differences between two software cases without any edit log. Eventually, an adequate method for computing closures and slices has to be defined. The decision on these issues is ongoing work.

(references removed)