Semi-automated Traceability Maintenance: An Architectural Overview of *traceMaintainer*

Patrick Mäder[1], Orlena Gotel[2] and Ilka Philippow[1]

[1]Department of Software Systems
Ilmenau Technical University, Germany
patrick.maeder|ilka.philippow@tu-ilmenau.de

[2]Department of Computer Science
Pace University, New York, USA
ogotel@pace.edu

Abstract

traceMaintainer is a tool that supports an approach for maintaining post-requirements traceability relations after changes have been made to traced model elements. The update of traceability relations is based upon predefined rules, where each rule is intended to recognize a development activity applied to a model element. Little manual effort or interaction with the developer is required. traceMaintainer can currently be used with a number of commercial software development tools and enables the update of traceability relations stored within these tools. This paper provides an overview of traceMaintainer's architecture and major components.

Keywords: Traceability, development activity recognition, rule-based traceability maintenance.

1 Introduction

traceMaintainer supports the (semi-)automated update of traceability relations between requirements, analysis and design models of software systems expressed in UML ([2], [1]). It can update traceability relations with little manual effort, a commonly cited barrier to traceability use within industry. This is possible by analyzing elementary change events that have been captured while working within a third-party UML modeling tool. Within the captured flow of events, development activities comprised of several events are recognized. These development activities are expressed as predefined rules. Rules consist of masks requiring an event with certain properties. Once recognized, the corresponding rule gives a directive to update impacted traceability relations to restore consistency.

The traceMaintainer prototype consists of several components and is designed with the objective of creating an implementation that is as independent as possible from specific CASE tools. The central component, the *rule engine*, handles the activity recognition and computes maintenance directives. It provides an interface for receiving new change events and requires an interface for querying and updating traceability relations. The change event interface will be used by a tool-specific *event generator* that recognizes changes to model elements and collects data about the changed element in order to create change events for the rule engine. The required query and update interface also has to be implemented by a tool-specific adapter. Depending on where the traceability relations are stored, the adapter gives access to relations stored within the model or to an external relationship repository such as EXTESSY ToolNET. In the prototype, we have implemented our own *traceSTORE* repository that stores relations within the tool, but provides significantly more functionality in terms of traceability than the base modeling tool itself.

In addition, a rule engine reads a *rule catalog* stored in XML format. This catalog can be edited and validated with a specific *rule editor*. Each of the major components is described further in this paper.

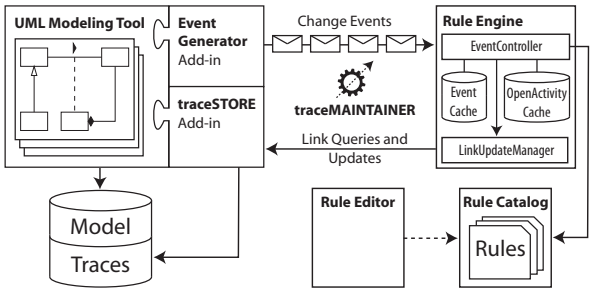

Figure 1. Overview of the traceMaintainer components

2 Rule Engine

The rule engine is the main component of traceMaintainer. It consists of an *EventCache* that holds a number of last incoming change events from the event generator, an *OpenActivityCache* that holds all partly recognized activi-

ties for the events currently held in the EventCache and a *link update manager* that determines the necessary traceability maintenance actions for recognized development activities. It depends upon the rule catalog for its operation (see Section 4).

The rule engine has a single user interface intended for the normal user. This is the interaction dialog that automatically pops-up in situations where a development activity has been recognized, but not enough information is available to carry out the traceability update completely automatically (see Figure 2). The dialog provides detailed information about the recognized development activity and the necessary update.

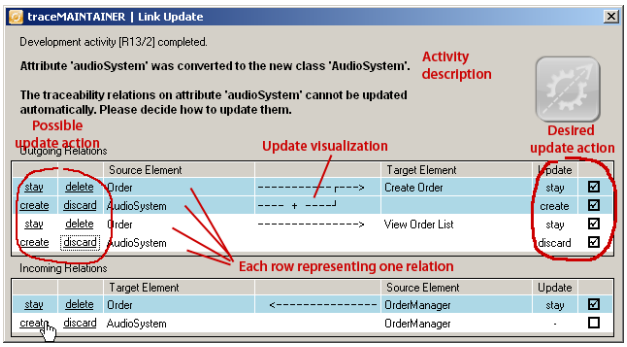

Figure 2. The user interaction dialog is displayed in situations where the update cannot be carried out automatically

The dialog shows two list boxes separating the incoming and outgoing traceability relations involved in the update context. Each row in these boxes represents an existing or potential new traceability relation. The user can decide to keep (stay is the default and preselected action) or delete existing relations on the source elements. For the evolved elements, the user can decide to create or discard the relation. A decision on the proposed relations without preselected actions determined is required to be able to complete the update, while a change of a preselected action is made possible in other cases but not required.

3 Event Generator

The event generator is created as an add-in to Enterprise Architect. It observes changes to elements of interest and captures a number of properties to the changed element. The types of elements to be observed and their properties of interest are defined within an accompanying information model stored in XMI format. The information model can easily be opened as a regular model in Enterprise Architect to allow the user to customize the generated events in terms of observed elements and collected properties.

4 Rule Editor and Rule Catalog

The rule editor is a stand-alone application that is intended to help in two usage scenarios. First, it validates an existing rule catalog upon opening it according to four categories of possible failures. Second, it provides functionality to edit and create all parts of a rule catalog whilst also validating the changes. For rules, the update and the description provided within notifications can be defined (see Section 2). To support definition, typical values for each property can be stored in the information model and provided within a drop-down box in the rule editor. Each entry made for a property is syntax-checked and all defined references are checked for their existence within the alternative. Failures are indicated by colored values (red means syntax mistake and violet means reference mistake.

5 Status

traceMaintainer provides an extensive set of features for implementing and maintaining traceability between a broad spectrum of UML model element types. Its main component, the rule engine, has been implemented independent of a specific modeling tool and supplies a well-defined API. In this paper, we have described tool-specific extensions for the Enterprise Architect modeling tool to satisfy the API and also enhance its existing traceability functionality. Initial experimental results have been encouraging ([2], [1]) and further industrial case studies are planned.

Since the rules are likely to evolve, we have created a rule editor for their definition and validation. We are currently investigating how to semi-automatically define rules by observing a developer performing change activities in situ using a rule recorder. We are further investigating how to handle the undo function within third-party modeling tools effectively, whilst still recognizing development activities accurately.

Acknowledgments The authors would like to thank Tobias Kuschke, Christian Kittler and Arne Roßmanith for implementing the prototype.

References

[1] P. Mäder, O. Gotel, and I. Philippow. Enabling automated traceability maintenance by recognizing development activities applied to models. In *Proceedings of 23rd International Conference on Automated Software Engineering ASE2008*, L'Aquila, Italy, Sept. 2008.

[2] P. Mäder, O. Gotel, and I. Philippow. Rule-based maintenance of post-requirements traceability relations. In *Proceedings of 16th International Requirements Engineering Conference (RE'08)*, pages 23–32, Barcelona, Spain, Sept. 2008.