Dynamic Analysis for Model Integration (Extended Abstract)

Andreas Fuhr, Tassilo Horn, Volker Riediger
University of Koblenz-Landau
{afuhr,horn,riediger}@uni-koblenz.de

Abstract

In reengineering, legacy systems have to be analyzed from different viewpoints. Models dealing with certain aspects of a system are used for that purpose, most of which are generated by fact extractors and parsers.

Since all of those models describe certain aspects of the same system, they have to be integrated with each other to allow a comprehensive analysis of the system as a whole.

This paper presents an approach using dynamic analysis to identify and manifest relationships between static model elements. It exemplifies the approach by integrating a business process model with a source code model.

1 Introduction

In reengineering, a comprehensive understanding of legacy systems has to be gathered. For that purpose, a consistent model dealing with all relevant aspects has to be created by integrating a multitude of different models generated by fact extractors, parsers or created manually.

In the SOAMIG project[1], a model-driven migration approach towards SOAs, including supporting tools and techniques, is being developed.

With SOA, there is an emphasis on business processes. By interviewing the developers and users of the system to be migrated, a business process model is created. It encompasses all functionality provided by the system from a subject-specific logical viewpoint.

Legacy source code is another central model. There are no links between the code and process model, but when migrating to services, it is crucial to be able to assign parts of the legacy system's code to activities in the business process model, in order to find service candidates.

It is hard to integrate these models using static analysis, because no uniform conventions are applied in both models which a heuristic approach could exploit. So dynamic analysis was chosen to map business processes to code realizing them. A similar approach using dynamic analysis for feature identification has already been described in [2]. In contrast to that approach, this paper will focus on mapping business processes to code.

2 The SOAMIG Repository

In the SOAMIG project, all models are contained in a central fact repository managed by two backends: the TGraph library *JGraLab*[2] and the *Flow Graph Manipulator* by *pro et con*[3]. In this paper, the focus is on the former.

TGraphs are directed, typed, attributed and ordered graphs, which proven their applicability in several reengineering projects [1]. The graph library *JGraLab* provides an efficient API for accessing and manipulating TGraphs, including the powerful query language GReQL.

Each TGraph conforms to a metamodel of a class of TGraphs. In the project, a metamodel encompassing all relevant aspects of the system has been created. This paper focuses on two main parts: legacy Java source code and business processes.

Java Source Code. The legacy system is written in Java, so the overall metamodel includes a part describing the complete abstract syntax of Java. The project partner *pro et con* provides a parser which generates TGraphs conforming to that metamodel out of source, class and jar files.

Business Processes. The business process part of the integrated metamodel is defined by a subset of UML activity diagrams. A converter for transforming activity diagrams exported as XMI [3] into a TGraph conforming to the integrated metamodel has been developed.

Traceability Links. The integrated metamodel introduces several traceability link classes defined between "activity" from the business process part and "method" and "class" in the Java part.

The task for the dynamic analysis is to create traceability link instances connecting activities in the static business process model with methods and classes in the static source code model.

3 Dynamic Analysis

The setting of the dynamic analysis is depicted in Figure 1. The SOAMIG Repository is a TGraph conforming to the

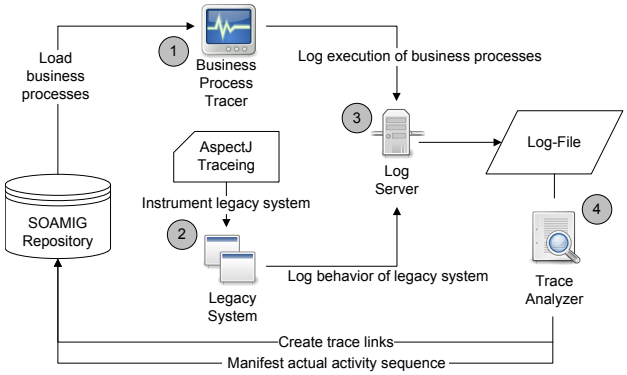

Figure 1: The setting of the dynamic analysis

integrated metamodel. It contains two isolated static models: the business process model which was created manually and the source code model generated by a parser. However, both models form completely isolated partitions in the graph. These are then integrated by connecting their

[1] SOAMIG (http://www.soamig.de) is funded by the Ministry of Education and Research (01IS09017C)

[2] http://jgralab.uni-koblenz.de [3] http://www.proetcon.de

elements using traceability links. For that purpose, a *Business Process Tracer* (1) is run in parallel to the instrumented *Legacy System* (2). Both components emit messages to a *Log Server* (3) that collects and synchronizes the messages and stores them in *Log-Files*. These files are then processed by the *Trace Analyzer* (4) creating traceability links between the elements of the static models in the SOAMIG Repository.

The four major components are explained in the following paragraphs.

The Business Process Tracer. The Business Process Tracer is a graphical tool developed during the project, which reads the modeled business processes from the SOAMIG Repository and visualizes them as shown in Figure 2.

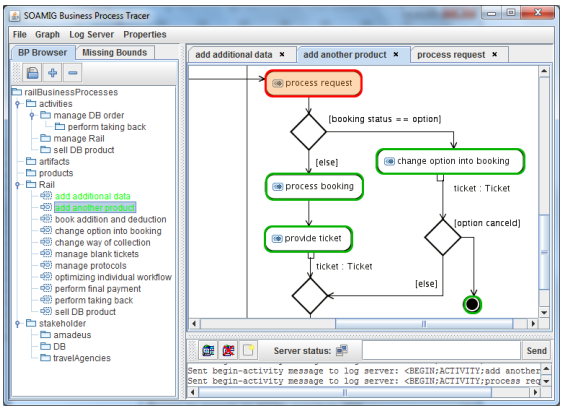

Figure 2: The Business Process Tracer tool

During execution, a user operates the tracer in parallel to the legacy system. In the tracer, an activity is selected and activated ("process request" in the screenshot) and then the user requests the functionality from the legacy system by using its user interface.

Whenever an activity is activated or deactivated in the tracer, a message containing a timestamp and that activity's name is sent to the Log Server.

First tests have shown that each modeled business process can be executed in multiple ways when using the current user interface of the legacy system. Therefore, the tracer does not enforce that the order in which activities are activated matches the order modeled in the idealized business process model.

The Instrumented Legacy System. The legacy system, consisting of a client and a server, is instrumented using AspectJ. The relevant aspects define pointcuts for the execution of methods and constructors. Before and after each pointcut, a message is sent to the Log Server. This message contains a timestamp, the name of the current thread and the qualified name and signature of the method or constructor that has been invoked.

The Log Server. The Log Server receives the messages sent from the Business Process Tracer and the instrumented legacy system and stores them in log files. Thereby, it provides a synchronization mechanism which guarantees the correct order of the logged messages, even though the legacy server, the client and the tracer are running on different machines with clocks ticking asynchronously.

The Trace Analyzer. So far, all components of the dynamic analysis framework dealt with recording the sequence of activity (de)activations in an executed business process and the corresponding low-level actions in terms of method and constructor calls, which were executed in the legacy system in response to that.

The Trace Analyzer's job is to extract meaningful information from the log files generated by the dynamic analysis framework and to integrate them back into the static models in the SOAMIG repository.

It iterates over the log file entries and from the begin/end-activity entries, it determines the activity in the business process model, which was executed at that time. Between the begin and end entry for an activity, there are method and constructor calls, which were executed in the legacy system while executing this activity. Therefore, a traceability link is created connecting the activity in the business process model to the definitions of the called methods in the source code model. In case of constructor calls, the link leads to the class definitions from which objects are instantiated.

Additionally, the actual sequence of executed activities is integrated into the business process model by creating special edges between activities. Each path through the graph consisting of activities and those edges represents one concrete instances of a business process (a *scenario*).

After integrating the static business process model with the static source code model by manifesting dynamically gathered information as edges in the repository, static analysis like model querying can exploit these enhancements. For example, a query can calculate all the classes and methods instantiated and called in the execution of a given activity including the dependencies. When trying to provide some technical activity as a service in a SOA by reusing legacy code, this is very important information.

4 Conclusion and Future Work

In this paper, dynamic analysis has been used to integrate a static business process model with a static source code model. The approach has been applied successfully during the SOAMIG project. Queries on the model integrated by the traceability links can now calculate comprehensive, global information. Depending on the project needs, the approach will be extended to several other aspects like architecture integration.

References

[1] J. Ebert, V. Riediger, and A. Winter. Graph Technology in Reverse Engineering: The TGraph Approach. In *10th Workshop Software Reengineering*, volume 126 of *GI-Edition Proceedings*, Bonn, 2008. Ges. f. Informatik.

[2] J. Quante and R. Koschke. Dynamic Object Process Graphs. In *European Conference on Software Maintenance and Reengineering*, pages 81–90, 2006.

[3] OMG. MOF 2.0 / XMI Mapping Specification, v2.1.1, 2007.