Towards Generic Services for Software Reengineering

Andreas Fuhr, Volker Riediger, Jürgen Ebert
Institute for Software Technology, University of Koblenz-Landau
{afuhr|riediger|ebert}@uni-koblenz.de

Abstract

Different software reengineering projects often perform similar reengineering tasks. This paper presents an industrial case study about an architecture recovery of a batch system using generic reengineering services. The case study is evaluated to identify key concerns for a generic approach.

1 Introduction

Reengineering projects often require performing similar tasks (e.g., dataflow analysis) on different artifacts like source code or models. Often, software analysis tools support certain technologies or programming languages only and are not applicable to other artifact languages. Providing reengineering tasks as orchestrateable services, implemented generically and mappable to specific technologies, enables a broader use of reengineering skills and supports reusing tools and techniques.

The case study presented in section 3 was part of the *Cobus* project. Cobus aims at recovering the architecture of a large COBOL system developed by Debeka, a German insurance company. Many of the tasks in that project were realized by means of generic reengineering services.

2 Context: Generic Reengineering Services

Generic reengineering services are services [Bel08] that aim at supporting reengineering tasks occurring in many reengineering projects (e.g., dataflow analysis). Generic reengineering services are implemented independent of programming languages or technologies of the subject system.

Input and output data of generic services are formally specified by generic metamodels. To apply generic services to specific artifacts, a compatibility concept is required, "translating" specific models of artifacts into models conforming to the generic metamodels of the services.

3 Case Study: Architecture Recovery of an Insurance Batch System

As part of the Cobus project, the architecture of Debeka's core system was recovered. 11 architectural viewpoints were specified by using a scenario-based approach [Fuh13]. For each viewpoint, requirements and domain knowledge were captured and metamodels specifying the viewpoints were developed. Subsequently, reengineering services for extracting and integrating data, for analysis and visualization were identified and implemented.

Figure 1 shows the orchestration of services to obtain the so-called *jobplan overview* and *jobplan execution* view[1]. The process includes extracting static and dynamic information ((1); scheduler source files and logs of the scheduler system and script executions) and integrating this information into one model (2). Controlflow ((3); computed on jobplan dependency graphs) and dataflow (3) within and between scripts are computed. Dataflow information is lifted to jobplan level (5) and data is filtered for visualization purposes (6). Finally, data is exported for visualizing the two views (7). The development of the dataflow analysis service is described in the following in more detail.

To apply generic services to specific models, a "translation" between specific elements and generic elements is required. E.g., statements in programming languages or batch script activities in the jobplan world both conform to generic controlflow items. There are several possibilities to realize such a translation. In Cobus, translations were mainly realized by establishing a generalization hierarchy between specific and generic metamodel elements. In some cases, generalization could not be used as one generic metamodel element was represented by a *set* of specific metamodel elements. Here, persistent views on the specific models were realized.

Based on the generic controlflow graph, the dataflow was computed generically. Dataflow is determined by an approach computing so-called *in* and *out sets*[2] [Aho08]. In contrast to dataflows in programming languages, where a relatively small number of data items (variables) have to be considered, in the batch system a huge amount of files and databases is accessed. Dynamic analysis revealed more than 250.000 entities used by a single jobplan.

Hence, major efforts were made to optimize runtime and storage performance of the generic dataflow analysis service. Based on properties of the controlflow graph, the dataflow computation algorithm is adapted to provide optimal performance for different domains.

[1] In the insurance system, jobplans are declarative dependency specifications that control the scheduler software of the batch system.

[2] In and out sets contain information about what data is alive before and after entering a dataflow item.

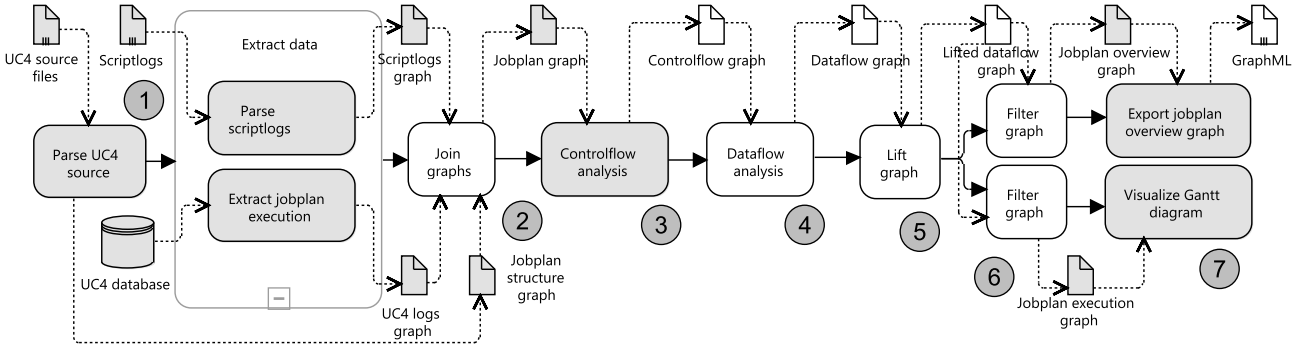

Figure 1: Orchestration of viewpoint-specific (gray elements) and generic (white elements) reengineering services for computing the jobplan overview and the jobplan execution views

As a result of the generic realization, dataflow computation will also be applicable to COBOL controlflow graphs computed on the structure of COBOL source files. Figure 2 shows an excerpt from the generic dataflow metamodel and mappings to specific metamodels for COBOL and jobplans.

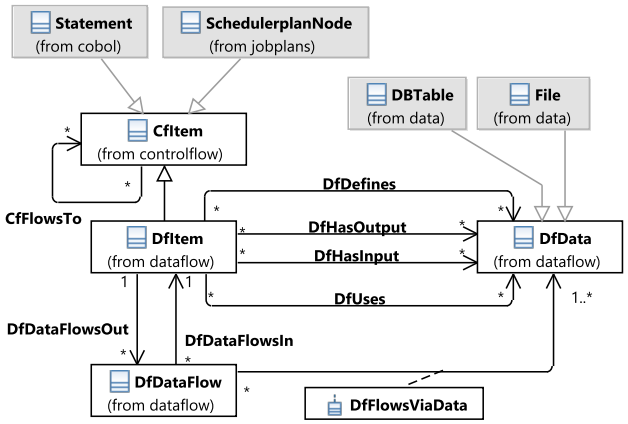

Figure 2: Generic dataflow metamodel and specific (gray) COBOL and jobplan metamodel elements connected by generalizations as compatibility concept

4 Problem Statement

When implementing architecture recovery processes in Cobus, several key concerns were identified that should be addressed in generic reengineering services research.

Generic services: Automated reengineering tasks are applied to artifacts specified and implemented by a broad range of technologies. Specification of generic metamodels, as well as definitions of interfaces and capabilities of services, are essential to enable for service orchestration. Also, a suitable component framework to technically integrate the services is required.

Compatibility concept: When services are implemented generically, specific data has to be "translated" to the generic input and output models. Such translations can be very complex. Introducing generalization dependencies between specific and generic metamodel elements may not be sufficient. E.g., a simple association in a generic metamodel can correspond to a complex path connecting many objects in a specific model. More sophisticated compatibility concepts, like view concepts or transformations are necessary.

Adaptivity: Depending on the domain that reengineering services are applied to, strategies about performant execution of algorithms may vary. Therefore, generic reengineering services should be self-adaptive in order to adapt to different domains seamlessly.

Process reuse: Reengineering activities comprise automated tasks and manual steps; both may reoccur in future projects. Manual steps as well as automated computational tasks – and therefore the complete reengineering process – should be addressed when thinking about genericity and reuse.

5 Conclusion and Future Work

In this paper, the recovery of a batch system's architecture was presented from the perspective of generic services research. First experiences in implementing generic services were provided and key concerns of a generic approach were stated.

The generic services used in the case study are still in a prototypical stage. Services are provided by plain Java classes and the orchestration is done manually. Though real-world problems can be solved efficiently, ongoing research is conducted to address more of the key concerns and to facilitate easy orchestration of the service landscape.

Acknowledgment: We would like to thank all members of the Cobus team for their valuable work.

References

[Aho08] A. V. Aho. *Compiler: Prinzipien, Techniken und Werkzeuge*. Informatik. Pearson Studium, München, 2 ed., 2008.

[Bel08] M. Bell. *Service-oriented modeling: Service analysis, design, and architecture*. John Wiley & Sons, Hoboken, 2008.

[Fuh13] A. Fuhr. Identifying Architectural Viewpoints: A Scenario-Based Approach. *ST-Trends*, 33(2):57–58, 2013.