Towards Reverse Engineering for Component-Based Systems with Domain Knowledge of the Technologies Used

Yves R. Schneider, Anne Koziolek
Karlsruhe Institute of Technology
{yves.schneider, anne.koziolek}@kit.edu

Abstract

Many developers today face the challenge of managing and maintaining existing legacy software systems. Improving the understanding of these systems is an important issue in addressing these challenges. To improve understanding, reverse engineering can be used to generate a higher-level representation. However, generic and extensible reverse engineering solutions that address multiple types of different technologies are missing or incomplete. This paper proposes to take a step in this direction. We describe the underlying idea of how used technologies such as frameworks and libraries induce parts of the architecture. Building on this, we describe our proposed approach of how the similarities of different technologies can be used to redevelop component-based architectures. By incorporating knowledge about technologies, we aim to improve the result of reverse engineering processes.

1 Introduction

Today, many companies rely on a variety of heterogeneous legacy systems. The management of software maintenance projects is generally a challenge. To perform these maintenance tasks, a developer must first understand the architecture of the system. One way to improve understanding is to automatically recover and create a model of the system attributes that are relevant to developers. For this purpose, techniques of model-driven reverse engineering can be used. Model-driven reverse engineering is the process of understanding software and creating a model, suitable for documentation, maintenance or reengineering [1].

In [4], Garcia et al. come to the conclusion that clustering of software entities is the almost uniformly applied method for the automated recovery of architectures. In most cases, a graph is generated on the basis of the source code, so that components can be reconstructed using clustering and/or pattern matching. Our new proposed approach, however, aims to use domain knowledge of libraries and frameworks for the reconstruction of architectural models from a system. Our assumption here is that components with their interfaces and their distribution are often explicitly determined by the technology used. By considering technologies we expect better results in reverse engineering, because heuristics like cohesion or coupling do not need to be used to determine components from the source code.

The remaining part of this paper is structured as follows: In Section 2, we first present the idea and the initial draft of our proposed reverse engineering approach. In Section 3, we discuss the related work and in Section 4 we draw our conclusions.

2 Approach

Software reuse is driven by the need to build systems that are more complex, reliable, cost-effective and delivered on time. This reuse of software can take place in many different ways, such as (re-) using libraries and frameworks. Both provide developers not only a setting to develop applications more effectively but can also induce parts of the software architecture.

The idea of our proposed approach is to model the knowledge about the domain of the technologies used in component-based software development in order to use them to reverse engineer the architecture from artifacts. This knowledge could describe, for example, how a component is implemented using a specific framework.

2.1 Technology Example

The following example briefly illustrates how we intend to use knowledge about a particular technology in order to automatically determine the interface of a component from the source code and thus automatically generate parts for a software architecture model.

Figure 1 shows a fragment of the source code for a REST endpoint that was implemented using JAX-RS. This endpoint is used to query and control an image provider service from the TeaStore case study [9]. JAX-RS is a Java programming language API specification that provides support for creating components that communicate via REST [6]. For this purpose several annotations are provided that help to assign a class as a REST endpoint. Knowledge about these annotations should be used to determine the interfaces of a component. The aim of our proposed approach is to map this technology-specific implementation of the interface into a software architecture model. Figure 2 shows an example of how this implementation of the interface can be mapped into a (UML-) model.

```
@Path("image")
public class ImageProviderEndpoint {
  @GET @Path("finished")
  public Response isFinished() {...}
  @GET @Path("state")
  public Response getState() {...}
  ...
}
```

Figure 1: Source code fragment of a REST endpoint to query a microservices [9].

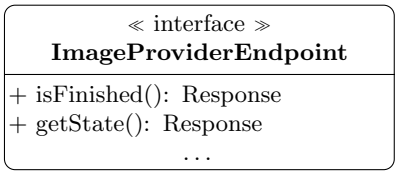

Figure 2: UML model for an interface.

2.2 Idea

The idea is to capture domain knowledge about technologies, i.e. as frameworks and libraries. This domain knowledge capture the effects of a deployed technology on the architecture of the system. We want to use this knowledge for reverse engineering of the architecture of a system using this technology. This process will automatically generate an architectural model of the system based on existing text-based artifacts. Here, text-based artifacts are considered that are written in the development of software system, e.g. source code or other configuration files such as deployment descriptors or project object models. In addition to text-based artifacts, domain knowledge about technologies is included as input. This domain knowledge is determined beforehand and independent of the system to be reverse-engineered. For this purpose, our approach includes a framework for model-driven reverse engineering, that should make it possible to capture this domain knowledge easily and reusably.

In order to achieve the reusability, we suggest making such common concepts in component-based software development explicit instead of implicitly coding them in recovery mechanisms. We achieve this by defining an additional concept metamodel with the relationships between the common concepts and the architecture model. These concept metamodels are general and widely used concepts that are used in component-based development. They describe the influences of the concept on a system architecture and how these are transformed into an architecture model.

2.3 Common Concepts Example

The following example briefly illustrates what such common concepts could be in the context of component-based software development and what could be the associated technologies. In Fig. 3, these concepts are shown as dashed boxes, and technologies that implement these concepts are shown as solid boxes. As an example, three different concepts are modeled in Fig. 3: component, container, and interface.

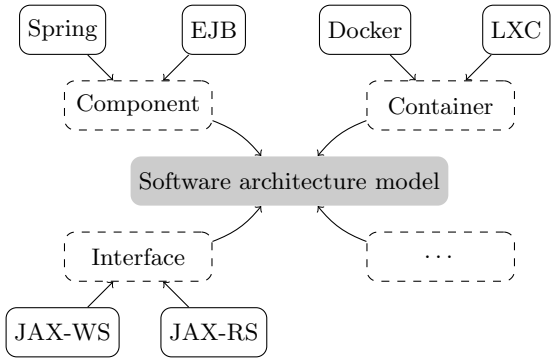

Figure 3: Excerpt of resemblances with the associated technologies.

The Spring Framework and the Enterprise JavaBeans (EJB) model are two technologies with which individual components can be easily implemented. The domain knowledge about these could be used to say how a component is implemented in the technology. The corresponding concept model in turn describes what a component in general is and how it is mapped into the architecture model.

Docker and Linux Containers (LXC) are technologies that isolate applications using container virtualization. Domain knowledge about them could be used to map the distribution of components to hardware resources. The corresponding concept model generally describes how a container is mapped into the model.

The Java API for RESTful and XML Web Services are technologies that describe component interfaces. Knowledge of these could be used to reverse engineer both offered and required interfaces from components. The associated concept model describes how interfaces of components are mapped into the model.

The concepts shown are just a possible excerpt for technologists and concepts to be supported by our reverse engineering approach. They are intended to illustrate how domain knowledge about specific technologies can be used to transform artifacts backwards into a software architecture model.

2.4 Benefits

Overall, the expected benefits of our proposed approach are improved understanding of the relationships between technologies and software architecture, simplified extensibility for new technologies, and improved results in reverse engineering a software architecture model.

By making the common concepts in component-based software explicit, we expect an improved understanding of the relationships between a technology and its underlying concept and the software architecture.

We expect this to reduce errors compared to direct transformations from software development artifacts

into a software architecture model. Our proposed approach can easily be extended to new technologies and concepts by introducing the additional concept model. When a new technology is added, it is only necessary to describe how it implements its associated concepts.

We expect improved reverse engineering results, since heuristics such as cohesion or coupling no longer need to be used to determine components from the source code. As we want to use knowledge about the technologies used, not least to detect components, we expect our proposed approach to generate models more in line with real architecture.

2.5 Realization

For our proposed approach we want to start with the Palladio Component Model (PCM) [7] as software architecture model. However, the concepts of our approach should also be transferable to other software architecture models.

In the first step, we plan to extract an Ecore-based model of the (Java-) source code with MoDisco [5]. We also want to use Ecore to describe our concept metamodels. The transformations from the code model to the concept metamodel and further to the software architecture model can be implemented as relationships in declarative transformation languages. These relationships model for a technology the knowledge of how to reverse engineer it from the code model.

3 Related Work

Bruneliere et al. developed MoDisco [5], a generic and extensible model-driven reverse engineering approach. MoDisco provides support for Java, JEE, and XML technologies to generate views of the architecture. While MoDisco is extensible with technologies, it does not support direct reuse of common concepts of a technology. In addition, the combination of multiple technologies to generate a view is not supported.

Krogmann introduces SoMoX [2], an approach for the reverse engineering of software component architectures. Based on an abstract syntax tree model, SoMoX uses heuristics and a graph-based hierarchical clustering approach to determine components. SoMoX does not use any special domain knowledge and therefore does not consider any technologies or other configuration files. Platenius et al. introduces Archimetrix [3], an architecture reverse engineering process for component-based systems. Archimetrix uses clustering techniques and also takes design deficits into account by providing the tools to detect and eliminate these defects. However, Archimetrix uses SoMoX for clustering and does not incorporate domain knowledge about the technologies being used.

Granchelli et al. introduces MicroART [8], a semi-automated architecture reverse engineering approach for Docker-based systems. They implement domain knowledge about specific technologies to redevelop the REST interfaces of a system. However, only components with such REST-based interfaces are captured and there are no starting points for adding more technologies.

4 Conclusions

In this paper, we proposed a model-driven reverse engineering approach that uses domain knowledge to recover architecture model form component-based software systems. The central idea of the proposed approach is to reverse engineer components with their interfaces and their distribution from existing software development artifacts such as source code or configuration files taking into account the technologies used. In order to be able to specify such domain knowledge for reverse engineering more easily and more extensively, we plan to introduced an additional abstraction level that describes how common software engineering concepts are mapped to component-based architectures.

We expect improved results in reverse engineering, since elements from the software architecture model will be explicitly determined by the technology used. We also expect good maintainability, since domain knowledge should be reusable and extensible, so that our approach can easily be extended to new technologies.

References

[1] S. Rugaber and K. Stirewalt. “Model-driven reverse engineering”. In: *IEEE Software* (2004).

[2] K. Krogmann. “Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis”. PhD thesis. KIT, 2010.

[3] M. C. Platenius et al. “Archimetrix: Improved Software Architecture Recovery in the Presence of Design Deficiencies”. In: *CSMR ’12*. 2012.

[4] J. Garcia et al. “A Comparative Analysis of Software Architecture Recovery Techniques”. In: *ASE’13*. 2013.

[5] H. Brunelière et al. “MoDisco: A model driven reverse engineering framework”. In: *IST* (2014).

[6] B. Burke. *RESTful Java with JAX-RS 2.0*. 2015.

[7] R. H. Reussner et al. *Modeling and simulating software architectures: The Palladio approach*. 2016.

[8] G. Granchelli et al. “MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems”. In: *ICSAW’17*. 2017.

[9] J. von Kistowski et al. “TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research”. In: *MASCOTS’18*. 2018.