Model-Driven Development of Resource-Oriented Applications

Silvia Schreier
Supervised by: Bernd Krämer

University of Hagen, 58084 Hagen, Germany
silvia.schreier@fernuni-hagen.de

Abstract. Resource-oriented applications are based on the architectural style Representational State Transfer (REST). Current frameworks support the implementation phase of REST applications, but offer no provisions for the analysis and design task. In addition, there are at most informal guidelines to accommodate REST constraints. This thesis suggests the model-driven development of resource-oriented applications that facilitates the observation of REST constraints and provides the advantages of model-driven approaches like formalized and platform-independent design solutions or early validation options. The current state of research and future research steps will be presented.

Keywords: REST, resource-oriented applications, web services, model-driven development.

1 Introduction

Representational State Transfer (REST), an architectural style that was introduced by Roy Fielding in 2000 [6], gains more attention particularly in the context of developing web based applications. The most important implementation of REST is the World Wide Web with the Hypertext Transfer Protocol and its other standards. Based on this ideas RESTful services [15] have evolved. They are an alternative to web services based on remote procedure calls (RPC) [14].

Resource-oriented applications are server applications offering an interface consisting of linked *resources*. A *resource* is identifiable and can be manipulated by *representations* using a *uniform interface*, which is defined by the protocol. The form of a *representation* is defined by a *media type*.

There are many technical solutions and frameworks, e. g., Jersey [2] and Webmachine [3], that support the implementation of such applications. However, only parts of the development process are usually supported by (semi-)formal models, e. g., documentation [5] and composition [13]. Particularly, for the earlier phases, like analysis and design, no suitable models for resource-oriented applications exist.

Vinoski [19,21,22] remarks the missing tool support and the lack of best practices as open issues in the development of resource-oriented applications. Beside, he mentions that most programmers are used to programming languages which use specific interfaces instead of a uniform one and that this is why many developers have objections [20].

To solve these issues, model-driven development (MDD), a generic term for techniques for creating software automatically from formal models, seems to be an appropriate paradigm [17]. The main advantage of MDD is the higher level of abstraction achieved, which allows designers to document, discuss and validate designs and apply patterns and best practices in a uniform and technology-independent way. Furthermore, it frees the developers from error-prone and tedious routine work. Because expert knowledge can be incorporated into the code generators, the source code and software architecture is uniform and of high quality. Even after changes generated documentation is up to date which supports the evolvability of resource-oriented applications. The key question of our research is whether methodologies and techniques of MDD can help to enhance the development and the understanding of resource-oriented applications.

After analyzing the research challenge in the next section, Sect. 3 describes the research plan of this thesis. We conclude in Sect. 4 with a summary.

2 Research Challenge

For historical and other reasons, RPC-based services are still dominating service-oriented applications even in areas that are well suited for a resource-oriented approach. Pautasso et al. [14] identify a set of problems in their architectural decisions based comparison of REST with WS-* services. The confusion about best practices and the missing standardization of design methods are mentioned as important weaknesses [14]. This confusion is based on the different ways RESTful services are implemented and which REST constraints they fulfill.

A lot of practical guides for developing RESTful services [15] and several frameworks have arisen. But best practices are described in a natural language and do not offer a common vocabulary. Because it is hard to develop and to discuss a design without more formal foundations, the first challenge is to identify a vocabulary for the different development phases and - based on that - a suitable metamodel.

A secondary research question we want to investigate is whether such a vocabulary can foster a better understanding of the foundations of resource-oriented, in contrast to operation-centric or object-oriented, design. Some existing work in the field of MDD for RESTful services has a too operation-centric view and therefore does not support a better understanding of the uniform interface [18]. The work on transformations from an operation-centric to a data-centric view of Laitkorpi et al. [11] should be taken into account when researching if such a metamodel can support the mapping from the application domain to an analysis model. The authors present a model-driven process for designing RESTful services, but skipped code generation. Instead of building a metamodel, which provides guidance, they base the identification of *resources* and the domain analysis on models and model transformation using the Unified Modeling Language (UML) and the Web Application Description Language (WADL) [8].

The existing implementation frameworks exhibit disadvantages because developers have to overcome the dichotomy of a resource-oriented application perspective and languages that promote specific interfaces and a more operation-centric perspective [20]. Furthermore, in the case of many frameworks the source code seems to contain several code duplicates, which typically causes copy and paste errors. This assumption should be explored in more detail by analyzing different implementations in different languages. If this holds true, it should be shown if code generation is possible and to which extent.

Particularly, in the field of documenting resource-oriented applications a lot of research has been done. Kopecký et al. [10] suggest microformats, which contain a model for RESTful services, but focus on documentation and discovery. With the same focus and also for composition Alarcón and Wilde [5] introduce a metamodel which is the basis for the Resource Linking Language (ReLL). The most important difference to WADL [8] and the microformats is that links are first class citizens in ReLL. Because currently the documentation has to be written manually, evolution often causes inconsistencies. To minimize the effort for creating and maintaining the documentation, the question if a (semi-)automatic generation for documentation is possible has to be answered.

Furthermore, the missing possibility to generate client stub code for RESTful applications is mentioned as another disadvantage of RESTful services by Pautasso et al. [14]. Another challenge is to show that it is possible to generate (parts of) client stubs based on a (semi-)formal model. A common use case is that existing legacy systems shall be extended by a resource-oriented interface. So the linking of such systems is another challenge. Having a formal metamodel, provides the opportunity to verify if a model satisfies desired properties. For example, it is necessary that the resources are linked to support hypermedia. Hence, it would be desirable that the graph built of all *resource types* and their links in between is (strongly) connected. Another interesting question is what properties can be verified with such a metamodel.

3 Research Plan

The first step on the research agenda is the development and evaluation of a metamodel. After finishing a first stable version, a textual and visual language for a better usability are planned as well as code, documentation, and client generation for different languages, frameworks, and formats. With these results a MDD of resource-oriented applications becomes possible. In addition to the tools, a development process needs to be defined for different scenarios, e. g., developing a new application or designing an interface for a legacy system. The approach and the prototypical implementations of the tools based on the Eclipse project [1] shall be tested with different case studies.

3.1 Defining a Metamodel

One of the most important aspects of this thesis is the development of the metamodel because its expressiveness determines the field of application addressed and the benefits of the metamodel. The version presented in [16] needs further refinements. Additionally, constraints and default values [17] need to be included.

To be able to model the functional aspects of a resource-oriented application for the entire development process, not only the structure but also the behavior needs to be taken into account [16]. So the metamodel consists of two parts, one for describing the structure, defining the *resource types* and the links in between, and one for describing the behavior. We suggest to use state machines to model the resource states and which methods are supported in each one. To describe the behavior of one method in more detail, imperative instructions are used.

The biggest challenges in this part is finding the proper elements and a suitable vocabulary on the one hand; on the other hand, we need to model *resource* manipulations that go beyond create, read, update and delete.

After the refinements the metamodel shall be applied to different case studies and compared to other models in this context [24]. Furthermore, it needs to be explored if one model can cover the requirements of all phases or if different models with transformations inbetween are more suitable.

Based on the chosen tools an automatic generation of a tree view editor is possible. Using such an editor is helpful and an alternative to describing the model using plain XML in a text editor. But a language with a concrete syntax which was designed for that domain is even better. This requirement leads to the next step of developing a textual and visual language.

3.2 Development of Textual and Visual Language

A metamodel is only the abstract basis for a framework for MDD. To get a useful domain-specific language (DSL) [7], only the concrete syntax needs to be added. Stahl et al. [17] mention that visual languages are better in illustrating structures and relationships. Additionally, they offer a better basis for discussions with bigger groups. If too many details are contained, a diagram becomes confusing for the reader. But for modeling a real world application the details need to be described as well. With textual languages defining details is more efficient. Because developers have different preferences regarding the kind of language, the goal is to develop both language types for this metamodel. For instance, the visual language could be used for the first analysis and identification of the different *resource types* and their linking as well as for showing the behavior of a *resource type* in terms of state machines.

The languages can be evaluated by testing their suitability for the documentation of best practices and patterns.

For developing easy to use languages, best practices and quality criteria for designing textual [7,23] and visual [12] languages should be taken into account.

3.3 Code Generation

As mentioned before, there are several frameworks in different languages for implementing resource-oriented applications. But these languages do not have *resources* or *resource types* as first class citizens. That is, why the different paradigms and abstractions become mixed.

If the metamodel allows us to build detailed models, we can generate code for different frameworks to reduce the amount of manually written code duplicates and to follow the *don't repeat yourself* (DRY) principle [9]. Probably, it is not possible to generate all necessary code, but avoiding repeating yourself is desirable. It is necessary to find the tradeoff between a model that overly detailed is hard to understand and a model that allows as much code generation as possible. The goal for this step is to provide code generators for at least one object-oriented framework, e. g., Jersey [2], and for one written in a functional language, e. g., the Erlang toolkit Webmachine [3]. The biggest challenge in code generation will be the different paradigms. This will show if the designed metamodel is independent enough from these paradigms.

Until now the concrete appearance of representations is not part of the model and only the contained data are described. For solving this issue, templates describing the concrete syntax can be used for representations sent by the server. For retrieving data from the receiving representations, a support for at least widely spread formats like XML and JSON should be developed to reduce the amount of manually written code again. For XML, e. g., XPath [4] could be used, similar to the selectors suggested in [5].

3.4 Documentation and Client Generation

There are several alternatives for documenting resource-oriented applications all having their advantages and disadvantages. The goal is to develop model transformations which allow us to generate WADL and ReLL documentations or at least a part of it out of the model. A first comparison of the current metamodel and the metamodels of WADL and ReLL shows that corresponding elements can be found. Hence, the development of such a transformation seems to be promising. The generation of (parts of) client stubs for different languages could be based on the documentation in WADL or ReLL to achieve a loose coupling from the server implementation.

4 Conclusion

Even though there is a lot of existing related work in developing resource-oriented applications no model-driven approach for the entire process exists. The goal of this thesis is to develop a metamodel for resource-oriented applications. Based on this theoretical work a framework for MDD of such applications is planned including the support for analysis and design as well as implementation and documentation. In case this can be achieved, it will be established that a metamodel can support the entire MDD process and what benefits and drawbacks it has. In some parts existing models can be used for comparison. Additionally, this thesis can benefit from a lot of best practices and tools in the field of developing frameworks for MDD.

References

1. Eclipse, http://www.eclipse.org/
2. Jersey, JAX-RS (JSR 311) Reference Implementation, http://jersey.java.net/
3. Webmachine, http://webmachine.basho.com/
4. XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/
5. Alarcón, R., Wilde, E.: RESTler: crawling RESTful services. In: Proceedings of the 19th International Conference on World Wide Web, pp. 1051–1052. ACM (2010)
6. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
7. Fowler, M.: Domain-specific languages. Addison-Wesley Professional (2010)
8. Hadley, M.: Web Application Description Language. World Wide Web Consortium Member Submission SUBM-wadl-20090831 (August 2009), http://www.w3.org/Submission/2009/SUBM-wadl-20090831/
9. Hunt, A., Thomas, D.: The Programatic Programmer. From journeyman to master. Addison-Wesley (1999)
10. Kopecký, J., Gomadam, K., Vitvar, T.: hRESTS: An HTML Microformat for Describing RESTful Web Services. In: 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pp. 619–625. IEEE (2008)
11. Laitkorpi, M., Selonen, P., Systa, T.: Towards a Model-Driven Process for Designing ReSTful Web Services. In: 2009 IEEE International Conference on Web Services, pp. 173–180. IEEE Computer Society (2009)
12. Moody, D.L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering 35, 756–779 (2009)
13. Pautasso, C.: Composing RESTful Services with JOpera. In: Bergel, A., Fabry, J. (eds.) SC 2009. LNCS, vol. 5634, pp. 142–159. Springer, Heidelberg (2009)
14. Pautasso, C., Zimmermann, O., Leymann, F.: Restful Web Services vs. "Big" web services: Making the Right Architectural Decision. In: Proceedings of the 17th International Conference on World Wide Web, pp. 805–814. ACM (2008)
15. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (2007)
16. Schreier, S.: Modeling RESTful applications. In: Proceedings of the Second International Workshop on RESTful Design, pp. 15–21. ACM (2011)
17. Stahl, T., Völter, M., Efftinge, S., Haase, A.: Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management. dpunkt.verlag (2007)
18. Valverde, F., Pastor, O.: Dealing with REST Services in Model-driven Web Engineering Methods. In: V Jornadas Científico-Técnicas en Servicios Web y SOA, JSWEB 2009 (2009)
19. Vinoski, S.: REST Eye for the SOA Guy. IEEE Internet Computing 11(1), 82–84 (2007)
20. Vinoski, S.: Demystifying RESTful Data Coupling. IEEE Internet Computing 12(2), 87–90 (2008)
21. Vinoski, S.: RESTful Web Services Development Checklist. IEEE Internet Computing 12(6), 94–96 (2008)
22. Vinoski, S.: RPC and REST: Dilemma, Disruption, and Displacement. IEEE Internet Computing 12(5), 92–95 (2008)
23. Völter, M.: MD*/DSL Best Practices (2011), http://voelter.de/data/pub/DSLBestPractices-2011Update.pdf
24. Wilde, E., Pautasso, C. (eds.): REST: From Research to Practice. Springer (2011)