Model-Based Service Discovery and Orchestration for OSLC Services in Tool Chains

Matthias Biehl[1], Wenqing Gu[1,2], and Frédéric Loiret[1]

[1] Royal Institute of Technology, Stockholm, Sweden
{biehl,floiret}@md.kth.se
[2] Ericsson AB, Kista, Sweden
wenqing.gu@ericsson.com

Abstract. Globally distributed development of complex systems relies on the use of sophisticated development tools but today the tools provide only limited possibilities for integration into seamless tool chains. If development tools could be integrated, development data could be exchanged and tracing across remotely located tools would be possible and would increase the efficiency of globally distributed development. We use a domain specific modeling language to describe tool chains as models on a high level of abstraction. We use model-driven technology to synthesize the implementation of a service-oriented wrapper for each development tool based on OSLC (Open Services for Lifecyle Collaboration) and the orchestration of the services exposed by development tools. The wrapper exposes both tool data and functionality as web services, enabling platform independent tool integration. The orchestration allows us to discover remote tools via their service wrapper, integrate them and check the correctness of the orchestration.

Keywords: Service Discovery, Service Orchestration, Model-driven Development, Tool Integration.

1 Introduction

Globally distributed software development teams need tool chains that are flexible, distributed and tailored to their development processes [12]. To deal with these new requirements, modern tool chains apply the principles of service-oriented computing [8,11], which deals with the generic integration of distributed services [9]. When applying the service-oriented principles to tool integration, tools expose both their data and functionality as services; these services are orchestrated to form a tool chain. The industry initiative Open Services for Lifecycle Integration (OSLC) [15], advocates a service-oriented, RESTful [7] architecture for managing tool data.

The challenge in adopting the OSLC approach for tool integration lies in finding appropriate mechanisms for discovering the RESTful services of remotely deployed development tools and to orchestrate the RESTful services of remote

M. Brambilla, T. Tokuda, and R. Tolksdorf (Eds.): ICWE 2012, LNCS 7387, pp. 283–290, 2012.

development tools. However, there is currently no standard and no practical support for discovering and orchestrating RESTful web services [17]. Due to the lack of a high-level design language for orchestration of RESTful web services, solutions are typically directly implemented in code; an overview of the details of this challenge is provided in [16]. As a result, the orchestration of tools requires a lot of manual work. In addition, inconsistencies can only be found on code-level, which is difficult, time-consuming and expensive.

In this paper, we propose a model-based approach to address both discovery and orchestration for RESTful services in the domain of tool integration. We introduce a domain-specific modeling language for tool integration that allows us to describe both the tool chain as an orchestration of tools and the specification of the services of each tool. This specification is the basis for both the discovery of tool services and the generation of an implementation. The domain specific model allows us to perform early correctness checks between the service usage and the service definition in the service specification.

2 Approach

To close the gap between discovery and orchestration of RESTful services for OSLC tool integration, our approach interleaves service discovery and service orchestration for tool integration, as illustrated in figure 1. We propose a model-based approach, which seamlessly integrates the results of service discovery with orchestration facilities. The pivot point of this approach is the discovered ToolAdapter metamodel; it is the central connection point between the service discovery and the service orchestration.

The automated process of service discovery is displayed on the vertical axis in figure 1 and explained in section 4. Discovery automatically deduces details

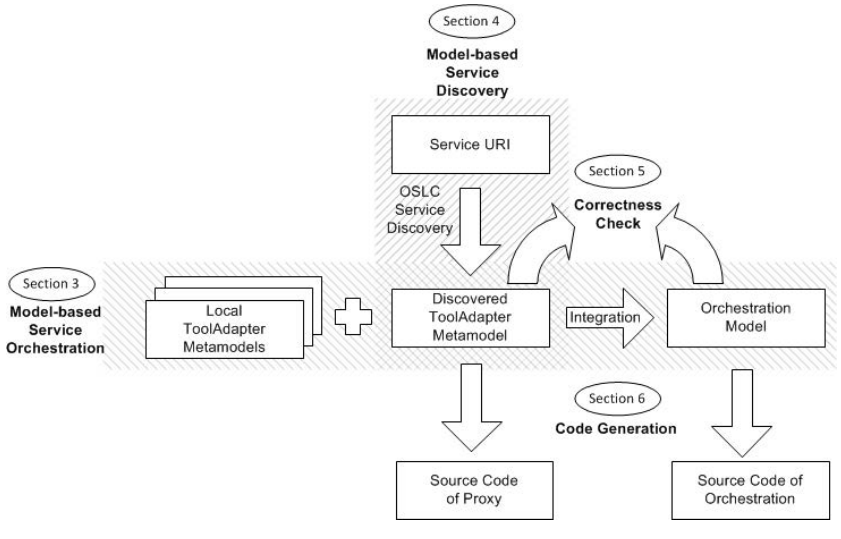

Fig. 1. Approach for model-based discovery and orchestration

of an already deployed tool adapter service, from which only a URL is known as an entry point. Discovery starts with the service URL, extracts the ToolAdapter metamodel using the OSLC *ServiceCatalogs* and *ServiceProviders* and finally generates code for the service proxies.

The process of service orchestration is displayed on the horizontal axis in figure 1 and explained in section 3. It starts with several ToolAdapter metamodels, which might be discovered or newly created and integrates the ToolAdapters into an orchestration model. The formalized ToolAdapter metamodel can even be used for verifying the discovered service definition against its usage in the orchestration model by a number of correctness checks, as described in section 5. Finally we generate code for the orchestration, as detailed in section 6.

3 Service Orchestration for Tool Integration with TIL

Tool chains are often put together in an ad-hoc manner. We promote a systematic development process, where a high-level design of the tool chain is created first. We would like to describe the design of a tool chain in such a way that all important design decisions of a tool chain can be reflected in it. This is why we apply the Tool Integration Language (TIL) [3], a domain specific modeling language for describing tool chains. TIL allows us not only to model a tool chain, but also to analyze it and generate code from it. Here we can only give a short overview of this language and for a detailed explanation of the semantics of TIL, we refer to [3].

In TIL, a tool chain is described in terms of *ToolAdapters* and the relation between them. For each tool, a *ToolAdapter* defines the set of data and functionality that is exposed by that tool in form of a ToolAdapter metamodel. The ToolAdapter metamodel is realized using EMF (Eclipse Modeling Framework)[1]. The relation between the ToolAdapters is realized by any of the following Channels: a *ControlChannel* describes a service call, a *DataChannel* describes data exchange or a *TraceChannel* describes the possibility to create traces. A trace is a link between two elements of tool data, which may reside in different tools. TIL offers three kinds of *ToolAdapters*. A *GeneratedToolAdapter* is newly created, locally deployed and the ToolAdapter metamodel is used as specification. A *BinaryToolAdapter* is included into the tool chain by locally deploying existing binaries and then binding to them. A *DiscoveredToolAdapter* is included into the tool chain by binding to an already deployed *ToolAdapter* on a remote server, it is merely specified by a URL. Realizing the binding in TIL requires a discovery process, which is described in section 4.

4 Service Discovery for Tool Integration

OSLC provides a catalog of linked metadata descriptions. The general idea is to use the catalog for remote discovery of tool adapters by following the links

[1] `http://www.eclipse.org/modeling/emf`

and parsing the metadata. We can discover the details of remotely deployed ToolAdapters that follow the OSLC specification. The key task of the discovery process is to interact with the OSLC directory services to extract a ToolAdapter metamodel. This ToolAdapter metamodel describes the data and functionality provided by the tool adapter and acts as an intermediate model in the discovery algorithm.

When parsing the OSLC metadata, we need to make some assumptions, because the OSLC catalog is not originally intended for the purpose of service discovery, but it contains useful information. The starting point for discovering services is the URI of the *ServiceProviderCatalog*. From the content of the *ServiceProviderCatalog* the algorithm follows to *ServiceProvider*s and *ResourceShape*s. All these resources are described in RDF [13] and we parse them with the Jena framework[2].

- **Step 1 - Parse ServiceProviderCatalog:** By parsing the response of an HTTP-GET on the URI of the *ServiceProviderCatalog*, we can identify a set of *ServiceProvider* resources. OSLC related information concerning the *ServiceProviderCatalog* is extracted and saved. For each *ServiceProvider* resource we continue with step 2.
- **Step 2 - Data or Control Service:** We perform a GET on the URIs of all *ServiceProviders*. We make the following assumptions for OSLC directory services: data and control services are encapsulated in separate *ServiceProvider* resources. For a control *ServiceProvider* only inlined *CreationFactory* resources exist, for a data *ServiceProvider* both *CreationFactory* and *QueryCapability* resources exist. With these assumptions we can identify the correct type of *ServiceProvider* by checking if the current *ServiceProvider* contains any *QueryCapability* resources. For a data *ServiceProvider* we proceed with step 3, for a control *ServiceProvider* we proceed with step 6.
- **Step 3 - Find Data Resources:** Each inlined *CreationFactory* or *QueryCapability* property represents one data resource, however, for the name of this data resource we have to assume it is contained in the URI of the inner property *resourceShape*, *resourceType*, *creation* or *queryBase*. With the listed sequence, we anticipate the resource name by extracting the last word of the URI. We use the simplest way to anticipate the correct name, which is to select the first URI in the listed sequence. For each data resource, we check if the URI of *ResourceShape* resource is given in the response. If it is provided, details of this data resource can be constructed by parsing the *ResourceShape* given, otherwise we query one or more specific objects of this data type to anticipate the structure of the resource. We follow step 4 if the URI of *ResourceShape* is given and step 5 otherwise.
- **Step 4 - Construct Data Resource Structures by Parsing the *ResourceShape*:** Properties of the current data resource are analyzed and added to the corresponding *class* in the ToolAdapter metamodel as *attribute*s or *reference*s. More specifically, for properties of primitive data types like *string*, *int*, etc. an *attribute* is added to the *class*. For other types we determine the referenced data type by analyzing the URI of the given *valueShape*

[2] `http://incubator.apache.org/jena/`

or *valueType* property as described in step 3 and add a corresponding *reference* to the current *class*. In OSLC, the multiplicity is described by the *occurs* property. Possible values are *Zero-or-one*, *Exactly-one*, *One-or-many* and *Zero-or-many*. After the structure of the data has been discovered, the ToolAdapter metamodel is updated.

- **Step 5 - Construct Data Resource Structures by Querying Object Details:** If there is no annotated *ResourceShape* given for a specific resource, we have to discover the structure of this resource by querying a specific instance of this data type. We obtain the list of objects by following the URI of *QueryCapability*. By analyzing the content of the response, we can obtain the attributes or references of the current data type. Since attributes are optional in RDF, we may need to query several objects, to increase the probability of acquiring a complete set of all the properties. We assume that the name of the attributes or references can be deduced from the local name in the properties of the response, and the referenced data type is directly from the local name of the resource type following the resource URI.
- **Step 6 - Find Control Resources:** By analyzing the URIs of the *creationFactory*, we can obtain a list of provided control resources. The resource name is obtained from the URI of *creation*.
- **Step 7 - Persist Discovered Tool Adapter Metamodel:** The discovery is finished and we save the discovered ToolAdapter metamodel.

5 Correctness Check

A correct and consistent TIL model is a prerequisite for the generation of correct source code that realizes the tool chain. We check the TIL model for correctness by analyzing if all service usages comply with their definitions. The definitions of the ToolAdapter services are located in the ToolAdapter metamodels. The usage of ToolAdapter services is specified in the TIL model, more specifically in the different types of Channels. The correctness check ensures that the usages of language concepts are conform to their definitions. The checks are performed early in the development process of a tool chain, on a model-level, before code is involved. Thus errors are relatively easy to detect and correct.

6 Code Generation

We describe the code generation in this section by describing (1) the chosen implementation framework, (2) the mapping of high-level concepts of TIL to the implementation and (3) the creation of proxies for DiscoveredToolAdapters.

Our approach is implemented using the Service Component Architecture (SCA) [2], a set of specifications for developing distributed Service-Oriented Architectures (SOA). SCA combines SOA principles [6] with principles of Component-Based Software Engineering (CBSE). While SOA provides the notion of loosely-coupled services, CBSE provides composability of software components.

SCA is a component model for implementing and composing heterogeneous services. We use the SCA implementation FraSCAti [19], which manages the web server infrastructure, produces the necessary glue code and also provides remote deployment, introspection and reconfiguration at runtime. SCA allows us to define RESTful services and bindings, which makes it possible to implement a tool chain according to OSLC. We found that SCA is an appropriate technology for realizing service-oriented tool chains based on OSLC.

The tool chain is an orchestration of services provided by both locally deployed *GeneratedToolAdapters* and remotely deployed *DiscoveredToolAdapters*. The latter are represented by local proxies bound to the remotely deployed tool adapter implementation. For implementing the interface we use the Service Component Architecture (SCA). By specifying the binding address of the remotely deployed ToolAdapter, SCA tool support can generate the proxy implementation that forwards calls to it. For each *DiscoveredToolAdapter* an SCA component is generated, acting as a local proxy of the discovered adapter. As a proxy, it provides the services of the ToolAdapter metamodel that were retrieved by the discovery process. The services of the local proxy are bound to the remote services provided by the remotely deployed ToolAdapters. SCA allows the specification of remote bindings that are managed transparently by the SCA runtime platform. The orchestration components are SCA components generated from the *ControlChannels* and the *DataChannels*, and bound to the proxy components according to the control and data flows they specify in the orchestration model.

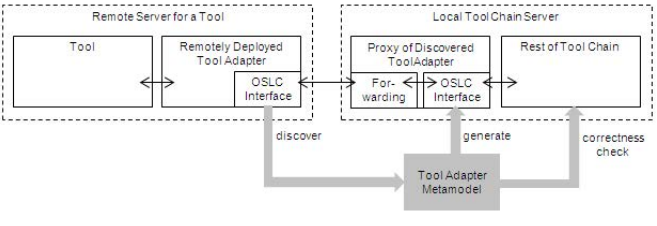

Fig. 2. Architecture of the Discovered ToolAdapter

We use the discovered ToolAdapter metamodel for generating code proxies for the ToolAdapter, which can be used to bind to the remotely deployed ToolAdapter instance. The complete ToolAdapter architecture is represented in figure 2. We distinguish between the remotely deployed ToolAdapter and the DiscoveredToolAdapter. The remotely deployed ToolAdapter already exists and is usually deployed on the same machine as the tool. In the implementation of the remotely deployed ToolAdapter, we separate the code that deals with the integration technology from the code that interacts with the tool. The external part of the remotely deployed ToolAdapter deals with the integration technology, the internal part interacts with the tool, e.g. via local APIs. The DiscoveredToolAdapter is a proxy to the remotely deployed ToolAdapter. It has the same interface as the external part of the remotely deployed ToolAdapter and its implementation merely forwards the service calls. Note that our approach is built

for the case in which we do not have access to the source code of the remotely deployed ToolAdapter. The benefits of automated generation are time, effort and cost saving. They can be achieved since the developers of the ToolAdapter do not need to learn the integration technology, nor do they need to implement any code that deals with the integration technology. A model-to-text transformation automatically generates the source code of the DiscoveredToolAdapter.

7 Related Work

Related work can be found in the areas of tool integration, service discovery and orchestration. We list the approaches by fields and point out approaches that are in the intersection of both fields.

Tool Integration: Model-based integration frameworks [1] use metamodeling for describing the tool data. However, these approaches provide neither concepts to model a complete tool chain nor concepts to describe the orchestration architecture of the tool chain. Model-based tool chains are usually realized locally. Tool chains based on the integration framework ModelBus [11] may be distributed. ModelBus uses the SOAP protocol, so discovery, orchestration and correctness checks can be performed.

Service Discovery and Orchestration: Web services based on SOAP [20] are usually described using WSDL (Web Service Description Language) [5]. WSDL is a W3C standard and is widely supported. In order to orchestrate WSDL-based web services, typically BPEL (Business Process Execution Language) [14] is used. The discovery and orchestration of RESTful web services is not equally well supported. The current BPEL 2.0 only supports WSDL 1.1, which is incompatible with RESTful services. RESTful web services can be described in WADL [10] and WSDL 2.0 [4], which is currently not supported by BPEL. Even if the next version of BPEL will support WSDL 2.0, a lot of manual work is required to consume the RESTful services provided, since the burden of creating the WSDL file has shifted from the service supplier to the BPEL designer. The reason is that no WSDL descriptions are provided by the RESTful service supplier. The main alternative is manual coding of the orchestration. A number of approaches for the orchestration of RESTful services have recently been proposed. The extension BPEL for REST [17] and the language Bite [18] have been developed for integration of RESTful services. In SCA, the binding of RESTful web services is possible, however a common Java interface must be used to invoke the web services. The added value of our approach is the domain specific support for OSLC, the correctness check of the orchestration and the code generation facilities.

8 Future Work and Conclusion

In the future we would like to improve the precision of the discovery algorithm and perform additional case studies of tool chains for different development processes. The cornerstone of this approach is the language TIL that describes both the orchestration of ToolAdapters and the ToolAdapter as models. The discovery

algorithm finds the details of an initially unknown ToolAdapter and represents them as a model. Both the orchestration and the results of the discovery are models, which allows us to verify their compatibility and correctness. As a consequence of this automated support for discovery, orchestration and correctness checks, distributed tool chains can be built faster and with less errors.

References

1. Amelunxen, C., Klar, F., Königs, A., Rötschke, T., Schürr, A.: Metamodel-based tool integration with MOFLON. In: ICSE 2008, pp. 807–810 (2008)
2. Beisiegel, M.: Service Component Architecture, Tech. Rep (November 2007)
3. Biehl, M., El-Khoury, J., Loiret, F., Törngren, M.: A domain specific language for generating tool integration solutions. In: MDTPI 2011 (June 2011)
4. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web services description language (WSDL) version 2.0 W3C, 26 (2007)
5. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web service definition language (WSDL). Technical report, W3C (March 2001)
6. Erl, T.: SOA Principles of Service Design. Prentice Hall (July 2007)
7. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
8. Frost, R.: Jazz and the Eclipse way of collaboration. IEEE Software (2007)
9. Gilmore, S., Gönczy, L., Koch, N., Mayer, P., Tribastone, M., Varró, D.: Non-functional properties in the MDD of SOS. In: SoSyM (2011)
10. Hadley, M.J.: Web application description language (WADL). W3C (2006)
11. Hein, C., Ritter, T., Wagner, M.: Model-Driven tool integration with ModelBus. In: Workshop Future Trends of Model-Driven Development (2009)
12. Herbsleb, J.D.: Global software engineering: The future of socio-technical coordination. In: FOSE 2007 (2007)
13. Klyne, G., Carroll, J.: RDF: Concepts and abstract syntax (2004)
14. OASIS. Web Services Business Process Execution Language, WSBPEL (2007)
15. OSLC Workgroup. OSLC Core Specification, version 2.0 (2010)
16. Pautasso, C.: On Composing RESTful Services. In: Software Service Engineering (2009)
17. Pautasso, C.: RESTful web service composition with BPEL for REST. Data Knowledge Engineering (2009)
18. Rosenberg, F., Curbera, F., Duftler, M.J., Khalaf, R.: Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. IEEE Internet Computing (2008)
19. Seinturier, L., Merle, P., Rouvoy, R., Romero, D., Schiavoni, V., Stefani, J.: A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures. In: Software: Practice and Experience (2011)
20. W3C. Simple Object Access Protocol (SOAP) 1.2. W3C (2007)