Towards Automatic Composition of Network Management Web Services

Torsten Klie
Technische Universität Braunschweig, Germany
Email: tklie@ibr.cs.tu-bs.de

Felix Gebhard
Exinit GmbH & Co KG, Germany
Email: fgebhard@exinit.de

Stefan Fischer
Universität zu Lübeck, Germany
Email: fischer@itm.uni-luebeck.de

***Abstract*— Despite the automation in many domains, network management still requires a lot of manual interaction. With the ongoing rapid growth of the Internet and the number of connected devices, the need for automated and intelligent management becomes even higher. Using Web services for management can be seen as a first step towards a better integration of business functions and technical management. However, offering basic management functions through Web services is still not enough for better automation. An autonomic management system is needed, which is able to perform more abstract tasks. These higher level tasks could be aggregated Web service calls. In this paper, we analyze existing techniques for automatic Web service composition and show how they can be used for network management. Moreover, we discuss our implementation of an automatic service composition engine.**

I. Introduction

The quick growth of the Internet and the number of connected devices stresses the need for further automation in the network management field. Currently, managing business networks requires a lot of manual work. Standard management technologies such as SNMP have a number of problematic issues [1]. One way to get rid of some of these issues is to use Web services, a technology that is independent from the application domain. Web services offer several benefits that make them very interesting for network management, e.g., using a general XML framework for service description and communication. Several Frameworks for Web services based management have been recently developed (e.g. OASIS WSDM, DMTF WS Management) and IETF NETCONF supports Web services as an optional feature.

To push automation, it will not be sufficient to set up management functions as Web services; additionally, searching and finding relevant services will be of significant importance. Only if a manager can find and call the relevant services without knowing many details of the underlying network real automation can be established. In other domains, especially in the area of commercial Web services, combined and composed Web services have shown advantages. Web services can be composed statically (which means manually in this case), but in recent years, several approaches for automatic Web service composition have been published.

This paper shows how the automation of network management can be supported by automatic Web service composition and is structured as follows: next, we will introduce Web service composition techniques briefly and will present existing composition approaches. Then we will describe the tasks an automatic composition engine must perform along with a management ontology in Section III. In Section IV we will present the design, some selected implementation details, and evaluation results of our composition engine. This paper will end with a brief summary and an outlook on further work.

II. Web service Composition

This section is a brief introduction to Web services composition including a comparison of different existing composition approaches: BPEL4WS, as a representative of template-based composition, and WSMO, WSDL- S, and OWL-S, which are semantic approaches.

A. Composition Basics

A composition consists of its synthesis and its orchestration. Synthesis means the way a composition is created. Possible synthesis models are manual composition, template based composition, and automatic composition.Orchestration is the execution of the composed service. It can be performed centralized or in a peer-to-peer fashion.

B. Template-based Composition

The XML-based Business Process Execution Language for Web Services (BPEL4WS) [2] has been designed to define processes that communicate with external Web services specified in WSDL 1.1 and present themselves as a Web service in WSDL 1.1. BPEL4WS compositions are complete executable process specifications which contain implementation logic. Parts of the process could be either basic activities (i.e. simple activities) or structured activities (i.e. a group of activities). These compositions are template-based compositions, so if a single Web service described in the execution plan is not available or produces an error, the whole execution fails.

C. Automatic Composition

In order to enable automatic compositions, requests and Web services must be stated in a way a machine can understand. The composition engine must be able to understand the semantics of the request to compose and execute an appropriate service. A request or task, respectively, could be described by specifying input and output parameters and the "world state" before and after the execution. These semantics can be expressed using an ontology that enables machines

to reason (i.e. draw conclusions) about described objects and relations.

An ontology is a knowledge representation of a specific domain in form of a data model which is expressed by an ontology language, a kind of description language. A common ontology language is the Web Ontology Language (OWL) [3]. With an OWL DL reasoner conclusions can be drawn and it is possible to infer new facts from existing ones. It can also be used to find inconsistencies within the ontology.

WSDL-S [4] is a simple extension to WSDL that allows for the description of IOPEs (functional semantic information: input and output parameters, preconditions, and effects) of Web services. Its advantages are the independence from ontology languages and the seamless integration into WSDL. However, some dimensions of a service composition, such as orchestration, choreography, and data transfer, are not definable in WSDL-S.

WSMO [5] is a complex conceptual model that describes all the dimensions of a service composition using ontologies. Requests (goals) and services are defined in the same way, but separately. Mediators can be defined to intercede between requests, Web services, and different ontologies. As of this writing, some core issues of WSMO, such as orchestration, choreography, data types and transfers, are not yet specified.

OWL-S [6] is a commonly known model for semantic description of Web services and Web service compositions. As opposed to WSMO, both requests and Web services are described via one element (ServiceProfile). A definition of mediators is not provided. Instead, they can either be emulated by plan generators (e.g. such as WGMediators) or be described directly as OWL-S descriptions (e.g. OOMediators). Due to the missing mediators, only static and template-based compositions can be described in OWL-S.

III. Automatic Composition of Management Web Services with OWL-S

In Section II, we have compared different Web service composition approaches. In the following, we will explain the major steps towards an automatic service composition with the OWL-S approach. We chose OWL-S because it allows for formal specification of compositions and it is not restricted to a specific architecture. Furthermore, there is a large number of tools and Java APIs available. The other approaches are not so useful for our purpose, because BPEL lacks semantic support, WSDL-S does not allow for the description of compositions, and WSMO is still missing important parts such as orchestration. However, a plan generator to implement automatic Web service composition with OWL-S is needed and descriptions of all semantic information relevant to the composition in form of an ontology must exist.

A. Building a Management Ontology

Different information modeling languages such as Structure of Management Information (SMI), Guidelines for the Definitions of Managed Objects (GDMO), and DMTFs Common Information Model (CIM) have been developed for network management. In order to enable machine-driven inferencing and reasoning, more components such as axioms and restrictions have to be defined. Several approaches [7], [8] have been made to translate existing models into a heavy-weight ontology.

In this work we want to focus on the aspects of service composition. Therefore, we do not address the problems related to ontology conversion towards a general (network) management ontology. Instead, we introduce our own simple but limited ontologies in OWL DL: a system ontology, a configuration ontology, and an operations ontology.

Fig. 1. Excerpt of the system ontology

Our system ontology contains the necessary elements to describe networks and individual devices. Each functionality in the network (switches, firewalls, Web server, printer, etc.) are represented by the concept `Role`. As shown in Figure 1[1], an individual of `Node` can have several roles, which have exactly one device type and one current configuration.Roles can have several addresses. Their state is represented by a `State` individual. `Node` individuals can be directly connected to other `Node` individuals.

The configuration ontology contains elements that can be used to describe configurations of devices. A configuration modelis necessary to base certain decisions on certain configuration characteristics. An individual of a `Configuration` consists of a base configuration and a device configuration. A `DeviceConfiguration` individual represents an installation of a foreign device at a Node individual such as a printer configuration of a network printer at a PC.

The operations ontology contains elements for the description and execution of actions. Basically, there are four different operations: `ReadConfiguration`, `ReadState`, `WriteConfiguration`, and a combined operation, which is used to classify operations. Combined operations are not

[1]The Figure has been generated using the Jambalaya plug-in for Protégé. The full OWL documents can be found in [9].

directly provided by devices, but can be executed as a composition of Web services or mediator services.

B. Tasks and Solutions

In [10], automatic Web service composition is divided into discovery, negotiation, composition, and invocation. In the following, we describe possible problems and solutions for each step.

1) *Discovery:* After a task has been specified, it is looked for Web services that can fulfill the task completely or in parts. The registry and the matching mechanism are the most important components here.
   a) *Registry:* The registry provides information about available Web services and how they are described. Unfortunately, UDDI does not provide sufficient information to evaluate automatically whether a Web service is suitable to fulfill a given task. On the one hand, the taxonomy provided is too simple and on the other hand, semantic information is missing. The OWL-S/UDDI Matchmaker [11] extends the taxonomies with semantic information and enables a machine to search for IOPEs. OWLS-MX [12] is another matchmaker, which contains a simple proprietary registry.
   b) *Matching:* The matching algorithm used by OWLS/UDDI matchmaker compares input- and output parameters of a request with the parameters of the listed Web services. The output parameters are compared first. If the degree of match between all output parameters of the request and a Web service is not "fail", the input parameters will be compared next.
   OWLS-MX has an extended matching algorithm and a slightly different result classification, which includes a syntactic similarity evaluation ($SIM(S, R) \geq \alpha$).
2) *Negotiation*: Before a found Web service can be used, it has to be negotiated how the service may be used (costs, legal restrictions, technical restrictions, etc.). In the network management context, this step is not so important, because the managed networks usually belong to a single business entity and the Web services manage individual devices.
3) *Composition*: In order to perform a task, several Web services, which have to be combined, might be necessary. This kind of composition can be called plan generation. Plan generation is a well-known subject of AI.
4) *Invocation*: This step deals with the correct execution of the Web services. This includes presenting the specified semantic data in a way which the particular communication protocol can handle.

IV. aNeMaC - Our Approach

The aNeMaC framework consists of a network management ontology presented in section III-A, some base Web services, e.g., to read and write configurations, and the base software aNeMaC that we describe in this section. Starting with the basic requirements, we will also discuss the most important design decisions and implementation issues. Finally, we will evaluate our solution and compare it to other existing solutions.

A. Requirements

It should be possible to describe tasks by specifying their input and output parameters. Furthermore, the administrator should be able to add descriptions of the devices present in the network. The core task of the system is the automatic plan generation to perform the specified tasks. Therefore, an evaluation of the available Web services which could perform sub-tasks as well as an arrangement of the Web services to be used w.r.t. order and data flows is done. If there are several possible plans, it must be possible to choose one, either manually or automatically. If there are no plans fitting the task description, the closest solution should be presented. Existing plans and task descriptions should be stored in a database. It should also be possible to modify the task descriptions later and to add, change, and remove OWL ontologies and Web service descriptions in OWL-S.

B. aNeMaC Architecture

The overall architecture of aNeMaC is divided into the four components management core, Web GUI, planning engine, and matchmaker.

1) *Management Core:* This core component provides the main functions to control and manage objects and to interact between components of the system. The main tasks of the management core are the management of task descriptions and plans, starting the plan generation process, as well as providing access to the database and OWL functions.
2) *Planning Engine:* This component contains functions for task description and plan generation. A `ServiceDescription` object should contain the task description provided by the administrator. The output of the planning engine is a `Plan` object that contains all the details of a generated plan. This includes the Web services, their parameters, and the mapping between the parameters of the used Web services and the task description. Such an object should describe a complete OWL-S composite process in order to enable an OWL-S import and export in the system.
3) *Web GUI:* This component allows to use aNeMaC via a Web browser in order to enable a decentralized use.
4) *Matchmaker:* The matchmaker component is called from the planning engine with a given service description and tries to find Web services that fit the requirements. In addition to parameter matching, a precondition test should be performed, if supported by the attached matchmaker. This component should be able to work with different existing matchmakers.

C. Implementation Details

A plan consists of a control flow and a data flow. The control flow in aNeMaC is composed of different Web Services, control structures with types like "Choice", "IfThen" or "RepeatWhile_End" and others[2] and preconditions. The data flow is expressed by a parameter mapping. A complete parameter mapping contains source parameters for all input parameters of the Web services and all output parameters of the task description. These source parameters could be output parameters of a Web service or input parameters of the task description.

One important detail of aNeMaC to point out is the plan generation, therefore a simple algorithm has been developed: Starting from the task description, matching and preconditions fulfilling Web services are searched by the matchmaker component. For each found Web service, a new `Plan` that contains this Web service as first service will be generated. Then it is checked whether the plan is complete. For each incomplete plan, additional Web services are searched. A list of Web services that provide output parameter for other not yet mapped parameters is generated for each incomplete plan. For each found Web service a copy of the original plan is created to which the Web service is added. Finally, it is checked whether the new plan is complete.

The generation of a parameter mapping, which is called after adding a service, follows a simple principle. All input parameters of the Web service and all output parameters of the task description that still have to mapped are queried. Then all available output parameters of the Web services and all input parameters of the task description are queried. For each parameter to be mapped, the set of available parameters is searched for a mapping. Therefore, it is checked whether the available parameters have the same concept or a sub-concept as the parameter to be mapped and whether those parameters will be available before the service with the parameters to mapped is called. If both conditions are true, those parameters will be mapped.

Plans are basically divided into complete plans, i.e. semantically matching task solutions, and incomplete plans, i.e. plans that lack parameters. A further classification is not needed here.

D. Evaluation

As the work is still in progress only a few performance evaluation results are available. These initial evaluations show that the amount of time used is growing quickly with the amount of available Web Services in networks: The need for checking for preconditions while rating Web Services is the crook, but there are a few options to optimize this slow behavior, e.g., caching mechanisms to keep old precondition checks or to adjust the use of the OWL-S MX matchmaker and the OWL-S API when checking preconditions.

[2]These mentioned types are borrowed from OWL.

V. Conclusion

In this paper, we have analyzed automatic Web service composition as a possibility to further automate network management. We have compared different basic Web service composition technologies and described aNeMaC, our approach of a composition engine for network management Web services.

Automatic Web service composition can be used to simplify complex network management tasks. aNeMaC with its network management ontology enables automatic composition that covers large parts of network management tasks. To use aNeMaC effectively in a production environment, several extensions and optimization should be made. First of all, a generic and network ontology that covers all aspects of networks should be developed and standardized. Then, all network resources should be accessible via Web services, either directly or via proxies. Furthermore, the management Web services of the resources will have to be described in OWL-S using the network ontology.

Besides optimization of aNeMaC, we are investigating, how Web service composition can be used as a complementary technique for policy refinement in policy-based management systems.

References

[1] J. Schönwälder, A. Pras, and J.-P. Martin-Flatin, "On the Future of Internet Management Technologies," *IEEE Communications Magazine*, vol. 41, no. 10, Oct. 2003.

[2] S. Thatte *et al.*, "Business Process Execution Language for Web Services – Version 1.1," Microsoft and others, OASIS Standard BPELv11-May052003, May 2003.

[3] P. Patel-Schneider, P. Hayes, and I. Horrocks, "OWL Web Ontology Language Semantics and Abstract Syntax," Bell Labs Research and University of West Florida and University of Manchester," W3C Recommendation, Feb. 2004.

[4] R. Akkiraju, J. Farrel, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma, "Web Service Semantics – WSDL-S," University of Georgia and IBM Corp., W3C Member Submission SUBM-WSDL-S-20051107, Nov. 2005.

[5] D. Roman, H. Lausen, and U. Keller, "Web Service Modeling Ontology (WSMO)," WSMO, WSMO Working Draft D2v1.2., Apr. 2005.

[6] D. Martin *et al.*, "OWL-S: Semantic Markup for Web Services," SRI International and others, DAML White Paper Release 1.1, Nov. 2004.

[7] D. Heimbigner, "DMTF - CIM to OWL: A Case Study in Ontology Conversion," in *Proc. 16th International Confernece of Software Engineering & Knowledge Engineering (SEKE)*, Alberta, Canada, June 2004.

[8] J. E. L. de Vergara, V. A. Villagrá, and J. Berrocal, "Applying the Web Ontology Language to Managament Information Definition," *IEEE Communications Magazine*, vol. 42, no. 7, July 2004.

[9] F. Gebhard, "Automatische Komposition von Web-Services für das Netzwerkmanagement," Master's thesis, TU Braunschweig, June 2006.

[10] A. Polleres, H. Lausen, and R. Lara, "Semantische Beschreibung von Web Services," in *Semantic Web - Wege zur vernetzten Wissensgesellschaft*, T. Pelegrini and A. Blumauer, Eds. Springer, 2006.

[11] N. Srinivasan, M. Paolucci, and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput," in *Proc. 1st Int. Workshop on Sematic Web Services and Web Process Composition (SWSWPC)*, San Diego, USA, July 2004.

[12] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX," in *Proc. 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Hakodate, Japan, Mai 2006.