The Role of Local Knowledge in Complex Web Service Reconfiguration

Sander van Splunter, Pieter H.G. van Langen, and Frances M.T. Brazier
Vrije Universiteit Amsterdam, Department of Computer Science, IIDS Group
{svsplun, langen, frances}@cs.vu.nl

Abstract

As the number of web services in repositories on the World Wide Web increases so will the number of complex configurations of web services. However, as the World Wide Web is dynamic, web services will come and go, temporarily or for good. As a result, complex web service configurations will need to be reconfigured on demand.

To this purpose, complex web service configurations need to include local knowledge about (1) the function, structure and behaviour of each component in a configuration, and (2) the dependencies between components at each level of composition. Templates are proposed as a means to represent such knowledge. To illustrate the process of reconfiguration, an example is given of reconfiguration of a complex web service, for which a template is used specifying both types of local knowledge.

1. Introduction

A web service is a service that is accessible by means of messaging. The messages are formatted according to standard web protocols, notations, and naming conventions [1]. On the World Wide Web, more and more services can now be accessed as web services, for instance Amazon.com and Google.com. As a consequence, the need to develop new web services from scratch gradually diminishes: more and more hosts offer web services that can be (re)used by developers for their own purposes. Repositories such as UDDI [2] are growing fast, so it is clearly becoming more practical to implement complex web services as configurations of existing web services.

However, as the World Wide Web is dynamic, web services come and go, temporarily or for good, necessitating reconfiguration of complex web service configurations on demand. If the same functionality of the system as a whole is required, this functionality will need to be made explicit.

In this paper, our claim is that incorporating local knowledge into a complex web service configuration is necessary to support the process of automated reconfiguration.

This paper is organised as follows. Section 2 discusses related research. Section 3 describes the concept of local knowledge: knowledge of a component and knowledge of dependencies between components, represented in templates. Section 4 presents an example of the use of local knowledge to determine the effects of substituting a single web service in a complex configuration. Finally, Section 5 discusses the results of this paper.

2. Related Research on Web Service (Re-) Configuration

Web service (re-)configuration can be viewed from at least two perspectives [3]: (1) a more syntactic-oriented business process composition using standard technologies to describe these processes such as SOAP, UDDI and WSDL, and orchestration languages such as XLANG, WFSL, WSMF, and BPEL4WS, and (2) a more semantic approach extending the business process approach with explicit semantics to enable automation.

ReFFlow [4] is one of the approaches to automated Web service configuration, based on workflows. Templates in reFFlow [5] encapsulate reusable parts of a process model, specified in their unified process model [6]. Configurations are specified in single level parameterised web service-based process templates, abstracting from web service process definition languages. The need for annotation is identified but has as yet not been implemented.

The METEOR-S project also focuses on automated web service composition, in particular on Quality of Service (QoS). Single level semantic process templates [7] define configurations of web services with constraints for each individual web service using properties as name, textual description, QoS metrics, input and output parameters. The need for composable structures is identified but has, as yet, not been addressed.

Parametric design has been adapted for web service configuration [8], based on the work in the IBROW [9] project on problem solving methods. Configurations of web services are represented in fixed single level templates containing parameters with adjustable

values, within predefined ranges and dependencies between parameters and functionality.

3. Local Knowledge

This section first explains the use of the term local knowledge. It then proposes a structure with which such knowledge can be represented.

3.1. Concept

As stated above local knowledge specifies knowledge about (1) the function, structure and behaviour (e.g., [10]) of each component in a configuration, and (2) the dependencies between components at each level of composition. This definition relies on the principle of compositionality, which specifies that the properties of a complex system are determined by its structure and the properties of its constituents [11]. A strengthened and often presupposed notion, the principle of local compositionality, is that the properties of a complex system are determined by its immediate structure and the properties of its immediate constituents [11].

Our definition of local knowledge is based on the principle of local compositionality and reads as follows: local knowledge is knowledge about the properties of a complex system. This knowledge includes knowledge about the immediate structure of the system and knowledge about the properties of the system's immediate constituents and their dependencies.

3.2. Representation

Local knowledge about a complex web service is represented in template structures using DAML-S [12]. A template describes a single level composition of one or more abstract, annotated web services, their input and output relations, and the conditions for (possibly parallel) activation of these abstract web services. A slot specifies the required properties for each of the abstract web services for which it has been defined (e.g., pre-conditions and post-conditions of the abstract web service). An abstract web service is either a primitive or a composed web service, or a template. A multi-level web service configuration based on one or more templates therefore includes not only information about the individual abstract web services of which it is composed, but also information on the requirements each web service has to fulfil (specified by the requirements for each slot). The control structure of a template states the conditions of activation of the specified abstract web services.

The effect of replacing one component with another in a complex composed web service based on one or more templates can be determined by first determining the local effects of a substitution and then determining the effects one or more levels higher in a composition.

4. Example: Web Portal Reconfiguration

This section illustrates the use of local knowledge for reconfiguration for a specific application domain, namely web portal creation. The complex web service used in this section creates a browseable portal for bibliographic data stored in multiple BibTeX-files [12]. The portal recognises different syntactical specifications for the same author, like "Sander van Splunter" and "Splunter, S. van", grouping such references together as appropriate. Section 4.1 gives an overview of the initial configuration. Section 4.2 specifies properties of the initial configuration relevant for the adaptation described in Section 4.3, and Section 4.4 discusses the influences of the adaptation on properties of the whole configuration.

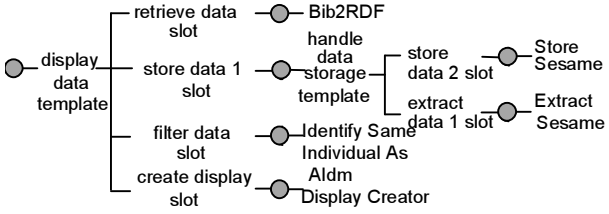

Figure 1. Composition of the complex web service for web portal creation.

4.1. Initial Configuration

Figure 1 shows the structure of the configuration of the complex web service for web portal creation. This configuration is based on two templates: *display data* with four slots, and *handle data storage* with two slots.

The *display data* template is the main template, and its four slots correspond to its four abstract web services: *retrieve data*, *store data 1*, *filter data* and *create display*. The primitive web services that have been placed in the *retrieve data*, *filter data* and *create display* slots of the template *display data*, are (see Figure 1):

- *Bib2RDF* retrieves a single BibTeX file and translates it to RDF statements.
- *Identify Same Individual As (SIA)* identifies same author names with different syntactical specifications, and denotes these as being the same by adding a Same Individual As tag.
- *AIdm DisplayCreator* creates a portal that displays the cooperation between different

authors. This service directly interacts with a Sesame repository, therefore the RDF data need not be extracted beforehand.

The *store data 1* slot contains the *handle data storage* template. This template, in turn, has two slots: *store data 2* and *extract data 1*. The primitive web services that have been placed in these slots are:

- *Store Sesame* stores RDF statements in a Sesame repository and handles Same Individual As tags.
- *Extract Sesame* retrieves all resource statements present in the Sesame repository and handles Same Individual As tags.

Given two sources of publication references, for example, the trace of activations of web services in the initial composition of the web portal reads as follows:

(1) Bib2RDF : Store Sesame
The first BibTeX file is retrieved, converted to RDF and stored in a Sesame repository

(2) Bib2RDF : Store Sesame
The same happens to the second BibTeX file.

(3) Extract Sesame : SIA : Store Sesame
After all data has been aggregated, all RDF statements on the references are extracted, and filtered on different syntactical specifications of authors, and the filtered result is stored again in the Sesame repository

(4) AIdm Display Creator
This web service accesses the Sesame repository and creates a portal on the available filtered information.

In this trace, 'A : B' denotes sequential activation of web services A and B including activation of the I/O stream of A to B. For further details on the portal creation scenario, see [12].

4.2. Requirements and Properties of the Initial Configuration

The *store data 1* slot in the template *display data*, specifies the following requirements. The abstract web service to be placed in this slot

1) must store and extract data (in this case RDF data);
2) needs to handles the identifiers created by the web service in the slot *filter data* (in this case identifiers created by the *Identify Same Individual As* service);
3) must take less than 5.0 ms to store a Kb;
4) must take less than 3.0 ms to extract a Kb.

The third and fourth requirements are behavioural requirements concerning the Quality of Service. The abstract services in the configuration shown in Figure 1 adheres to these requirements.

The properties of the template *handle data storage* as stated in its annotations fulfil theses requirements. The precise values of storage and execution depend on the template's constituents: more refined properties of storing depend on the services placed in the template's slots *store data 2* and *extract data 1*.

The properties of the web service *Store Sesame* placed in the *store data 2* slot are that it:
-provides storage functionality in a Sesame repository;
-handles Same Individual As tag;
-has an average estimated execution time of 1.0 ms/Kb.

The properties of the web service *Extract Sesame* placed in the *extract data 1* slot are that it:
-provides extraction functionality in a Sesame repository;
-handles Same Individual As tag;
-has an average estimated execution time of 0.8 ms/Kb.

4.3. New Adapted Configuration

The World Wide Web is a dynamic environment, in which these services come and go. To illustrate the implications of loss of access to a web service, the reconfiguration of the complex web service *handle data* storage, consisting of a template and two web services, described above, is depicted. A new web service or template is needed that fulfils the requirements posed by the *store data 1* slot of the *display data* template.

A new instantiated template is found: *handle data storage 2,* with three slots: *store data 3*, *extract data 2*, and *manipulate data*. These slots are filled respectively with the web services *Replace SIA Tag*, *Store Sesame Simple*, and *Extract Sesame Simple*. The configuration is illustrated in Figure 2.

The trace for storing data within the complex web service *handle store data 2* is more extensive than in the previous case. The trace reads as follows:

(1) Store Sesame Simple : Extract Sesame Simple :
Replace SIA Tag : Store Sesame Simple
RDF data on new references is stored for aggregation, and all data is extracted, including the data on the new reference. Multiple syntactical references to the same author are all replaced by one of these instances. Storing the modified data in the repository is the last step.

The trace for extracting data within the complex service *handle store data 2* is comparable to the previous configuration, namely activation of *Extract Sesame Simple.*

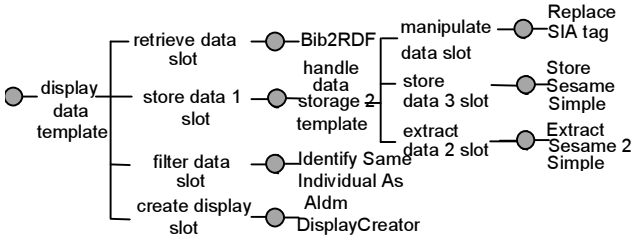

Figure 2. Composition of the adapted web service for web portal creation.

The properties of the web service *Store Sesame Simple* placed in the *store data 3* slot are that it:
-provides storage functionality in Sesame repository;
-has an average estimated execution time of 0.7 ms/Kb.

The properties of the web service *Extract Sesame Simple* placed in the *extract data 2* slot are that it:
-provides extraction functionality in Sesame repository;
-has an average estimated execution time of 0.6 ms/Kb.

The properties of the web service *Replace SIA Tag* in the *manipulate data* slot is that it:
- r:eplaces elements related to each other by a Same Individual;
-has an average estimated execution time of 1.1 ms/Kb.

4.4. Changes of the Properties of the Adapted Configuration

The properties of the template *handle data storage 2* fulfil the requirements set by the slot *store data 1* of the template *display data*. The extent to which it fulfils execution requirements, however, depends on local knowledge on the web services in the *handle data storage 2* template. This information is available at the level of the instantiated template, based on the following local information.

The execution time for extracting data depends only on the execution time specified for the primitive web service *Extract Sesame Simple*,: 0.6 ms/Kb (Note that this is well below the required 3.0 ms/Kb).

The execution time of storing data depends on all three constituents of *handle data storage 2* as described in the trace in the previous section. The execution time is the sum of the activation times of all services activations for storing RDF data: 0.7+0.6+1.1+0.7 = 3.1 ms/Kb. This is below the required 5.0 ms/Kb.

The example above illustrates an adaptation of a complex web service configuration, in which, from the perspective of the main template (*display data*), the behaviour changed, whilst the functionality remained the same. This process can be done fully automatically given the information specified as requirements for each of the slots, and as properties for each of the abstract web services.

5. Discussion

This paper presents an approach with which properties of complex web services can be defined locally and propagated to the level of the complex service itself. For each level in a multi-levelled web service configuration, properties of individual web services are specified together with the dependencies between web services at that level. Properties are propagated to a higher level if and when appropriate, making reconfiguration depend only on local requirements and properties.

Wyner and Lee [13] describe a related, limited concept of upward property propagation for specialisation hierarchies in process modelling (e.g., class hierarchies), in which the propagation is equal to the inverse of inheritance. If a property occurs in all specialisations of a superclass, then the property also holds for this superclass. Klas et al. [14] apply a similar concept of upward propagation, namely as inverse inheritance, to the creation of meta-classes over multiple classes in databases with different data models.

In parameterised web service configuration, Ten Teije et al. [8] propagate the effect of the change of a parameter/web service up to the level of the template. This approach does not, however, support combinations of complex web services.

The METEOR-S project supports a concept of upward propogation that is closely related to the concept presented in this paper. Given a web service configuration with atomic web services, they have devised an automated method to determine the aggregated values of attributes concerning QoS for the web service configuration as a whole [15]. A number of QoS dimensions (time, cost, fidelity, and reliability) are defined in enumerable functions and a computational model for QoS has been devised. Both the computational model and the constructs can be integrated in the templates described in this paper within a single level. Jaeger et al. [14] define constructs with which QoS aggregation over compositions of Web services can be specified, again providing a means to define behaviour within a template.

The use of local knowledge within templates to define properties of web service configurations can be used by human developers to reconfigure a complex web service when necessary. Templates can also be

used as a basis for automated reconfiguration providing local knowledge at each level in a multi-level composition. The authors have implemented a prototype system capable of reconfiguration using templates. More research is needed on ontologies with which properties can be expressed, and also on the most relevant functions on which upward propagation can be based.

Acknowledgements

The authors thank Debbie Richards from the Department of Computing at Macquarie University in Sydney, Australia, for her contributions to this research. Furthermore, the authors are grateful to the NLnet Foundation (www.nlnet.nl) for their support.

6. References

[1] Unified W3C Glossary and Dictionary, URL: www.w3.org/2003/glossary/subglossary/xkms2-req, 2003.

[2] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y.L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen, "UDDI version 3.0". Published specification, Oasis, 2002.

[3] B. Srivasta, and J. Koehler, "Web service composition – current solutions and open problems". In *ICAPS 2003 Workshop on Planning for Web Services*, 2003, pp. 28-35.

[4] D. Karastoyanova and A. Buchmann, "ReFFlow: a model and generic approach to flexibility of web service compositions", *International Conference on Information Integration and Web-based Applications and Service (iiWAS 2004)*, Jakarta, Indonesia, 2004.

[5] D. Karastoyanova and A. Buchmann, "Automating the development of web service compositions using templates" In *Proceedings of Geschäftsprozessorientierte Architekturen Workshop at Informatik 2004*, Germany, 2004.

[6] D. Karastoyanova, "A methodology for development of web service-based business processes". In *Proceedings of AWESOS 2004*, Monash University 2004.

[7] K. Sivashanmugam, J. Miller, A. Sheth, and K. Verma, "Framework for semantic web process composition", *Semantic Web Services and Their Role in Enterprise Application Integration and E-Commerce, Special Issue of the International Journal of Electronic Commerce (IJEC)*, C. Bussler, D. Fensel, N.Sadeh (Eds.), Vol. 9, No. 2, 2004.

[8] A. ten Teije, F. van Harmelen, B.J. Wielinga, "Configuration of web services as parametric design". E. Motta, N. Shadbolt, A. Stutt, N. Gibbins (Eds.): *Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004*, Proceedings. LNCS 3257, ISBN 3-540-23340-7, Springer, UK, 2004, pp. 321-336.

[9] V. R. Benjamins, E. Plaza, E. Motta, D. Fensel, R. Studer, B. Wielinga, G. Schreiber, Z. Zdrahal, and S. Decker: "An intelligent brokering service for knowledge-component reuse on the world-wide web". In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'98)*, Canada, April 18-23, 1998.

[10] J.S. Gero, "Design prototypes: a knowledge representation scheme for design", *AI Magazine* 11(4): 1990, pp. 26-36.

[11] Z.G. Szabó, "Compositionality, Stanford Encyclopedia of Philosophy"
URL: http://plato.stanford.edu/entries/compositionality/.

[12] D. Richards, S. van Splunter, F.M.T. Brazier, and M. Sabou, "Composing web services using an agent factory", in *Extending Web Services Technologies, The Use of Multi-Agent Approaches Series: Multiagent Systems, Artificial Societies, and Simulated Organizations*, ISBN: 0-387-23343-1, L. Cavedon, Z. Maamar, D. Martin, B. Benatallah, (Eds.) Vol. 13, 2005.

[13] G. Wyner, and J. Lee, "Applying specialization to process models". In *Proceedings of the Conference on Organizational Computing Systems*. Association for Computing Machinery, Milpitas, California, United States, 1995, pp. 290 - 301.

[14] W. Klas, E. J. Neuhold, M. Schrefl, "Metaclasses in VODAK and their application in database integration". *In: Arbeitspapiere der GMD - Gesellschaft für Mathematik und Datenverarbeitung*, No. 462, Germany, 1990.

[15] J. Cardoso, A.P. Sheth, J.A. Miller, J. Arnold, K.J. Kochut. "Modeling quality of service for workflows and web service processes", *Web Semantics Journal: Science, Services and Agents on the World Wide Web Journal*, 1(3), 2004, p. 281-308.

[16] M.C. Jaeger, G. Rojec-Goldmann and G. Mühl: "QoS aggregation in web service compositions". *The 2005 I5 International Conference on e-Technology, e-Commerce and e-Service (5-05)*, Hong Kong, 2005, pp. 181-185.