Value-Added Web Services Composition Using Automatic Program Synthesis*

Mihhail Matskin and Jinghai Rao

Department of Computer and Information Science
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
{misha,jinghai}@idi.ntnu.no

Abstract. The number of products and services available over the Internet increases dramatically during the last years and it is already beyond the human ability to analyze and combine them efficiently. At the same time if we consider works in software engineering (and, in particular, in component-based system development) then we can find quite strong similarity in problem description. The web services can be treated in a way similar to software components and service composition can be considered as problem of software synthesis and composition. Basic idea of our approach is applying the software synthesis and composition methods to value added web services composition. We also describe some technical details within the approach, in particular, a web service description language and a program synthesis method.

1 Introduction

The number of products and services available over the Internet increases dramatically during the last years and it is already beyond the human ability to analyze and combine them efficiently. In order to support a customer in such analysis, several product/price comparison systems are developed and they give the customer opportunity to compare prices for the same product from different suppliers [7]. Most of the tools compare only product prices while there are also some tools which perform a multi-criteria comparison [8] taking into account delivery conditions, guarantee etc. However, we think that problem of web service analysis and selection, in general, is more complex than selection and analysis of products. The complexity comes from the following two sources: first, it is not so easy to define and evaluate selection criteria for web services and, second, web services can be composed in order to satisfy customer's requests. Taking into account that the component services can be developed by different organizations, which provide different offers, the ability to efficient integration of possibly heterogeneous services on the Web becomes a complex problem (especially for dynamic composition during runtime [2]).

Currently, services composition is basically made by predefined workflow model or business logic. For example, in eFlow [5], a composite service is modeled

* Revised June, 2002 for inclusion in WES processdings

Ch. Bussler et al. (Eds.): WES 2002, LNCS 2512, pp. 213–224, 2002.

by a graph (the flow structure), which defines the order of execution among the nodes in the process. PPM[14] uses state machine to specify the possible states of a service and their transitions. Some XML-based languages, such as WSFL [9] and composite process in DAML-S [1] are also used to define the sequence and links among the services components.

In many ways, a composite service is similar to a workflow. For example, the provider should specify the flow of services/work items together with the control and data flow between the services/work items. However, workflow doesn't always provide an efficient solution for the problem of dynamic matchmaking and composition of web services [4]. In particular, there can be a large number of web service components with the same functionality and, even if the workflow has been predefined, we may still have problem to retrieve suitable (or the best) service components or dynamically combine them together in order to create a new service.

At the same time if we look at works in software engineering (in particular, in component-based system development) then we can find quite a strong similarity in problem description. In particular, the web services can be treated in a way similar to software components and service composition can be considered as problem of software synthesis and composition. Some methods of software components retrieval and composition were proposed in the Automated Software Engineering area [13,11]. Several software composition systems are designed based on these methods [16,3]. Main idea of these methods is that the components can be composed using their interface, pre-/post- conditions and reasoning methods (for example, deductive inference).

Basic idea of our approach is applying the software synthesis and composition methods to value-added web services composition. In order to do this the following facility programs are needed:

- A compiler which translates the web service description language, e.g. WSDL or DAML-S, into formal logic or other formal component description language.
- A synthesis mechanism which automatically selects, adapts and composes web service components.
- A manager that invokes the web service components and transfer data between them.

In this paper we propose an approach to using automatic program synthesis in composition of value added web services. Section 2 presents a working example. Then we describe some important technical details within the approach, in particular, the web service description language and a program synthesis method. Finally, we present conclusions and future work.

2 Working Example: Value-Added Web Services Composition

We consider a value-added web services composition for illustration of our approach to services composition.

Value-added services differ from core services - they are not a part of core services and they have unique properties/characteristics. In particular, they may stand alone in terms of operation and profitability as well as provide adds-on to core services. It is important to mention that value-added services may allow different their combination and they may provide incremental extension of core services.

As a working example we consider sport equipment selling web services. A core service, in this case, may receive sport equipment characteristics (such as measurements, brand, model etc) and it may provide prices, availability and other requested characteristics as output. We may assume that there can be several providers of sport equipment and prices for available equipment are compared by the web service.

In order to be more particular we consider a ski selling web service. We also assume that some web services consider measurements in inches and some in centimeters. Some services give prices in US dollars (USD) and some others in a local currency.

We can consider the following possible value-added services in our case:

- currency converter service - converts USD to other currency according to the current rates
- measurement converter - converts inches into centimeters and vice versa
- ski length selection service - provides a recommended ski length based on body measurements (in particular, based on height). We also assume that this service accepts body measurements as well as produces recommended ski length in centimeters.

The above-mentioned value-added services allow different their combination as well as combinations with the core services. In particular, measurements converters can be used both before and after ski length selection service depending on inputs. The converters can also be applied to the core service. Combination of value-added and different core services may give a large number of possible measurements conversions. In general case, number of possible combinations can be very high and it would be practical to configure them dynamically depending on input values. This may require a flexible configuration tools. There also exists a possible solution which provides converters to all inputs and outputs of all web services (both core and value added), however, this may cause a big overhead in service provision. Taking into account that each service may require payments for its usage such solution may not be acceptable in most of cases.

We also would like to mention that our working example is simplified and it is made more abstract than practical cases. This is done in order to keep simplicity of presentation. In a more practical case there can be more services available. In particular, there can be several recommendation services (recommendation based on skills, based on fashion, based on age etc.) which can be composed dynamically as well as payment and delivery services.

3 Some Issues of Services Presentation and Composition

3.1 Functionality and Non-functionality Conditions

Web services selection is an important step in web services composition. When we talk about services selection then we separate functionality and non-functionality conditions. The functionality conditions express a transformation performed by the service and they can be represented via the service input-output parameters. These conditions are basis for matchmaking process - selection of service by name, by what are its input arguments and by what are its output results. The non-functionality conditions provide additional information about the service and constraints on its usage (for example, information about service provider, cost, QoS, security, etc.)

It is quite common that several similar services can be selected by the same functionality conditions. Non-functionality conditions involve deeper level of services description and they may require manipulation with semantic descriptions, annotations etc.

Taking this into account we consider two levels of web services selection and composition:

- coarse-grain selection - selection which is based on functionality conditions and it is usually performed first
- fine-grain selection - it is based on non-functionality conditions and it can be considered as an optimization of coarse-grain selection

We also would like to notice that it may happen that in some cases coarse-grain selection is enough and no fine-grain selection is required. However, if more than one service is selected by the coarse-grain selection, then we can apply the fine-grain selection.

For the coarse-grain selection, we can apply both keywords matching and semantic matching for each input-output parameter. Since web services exist in an open environment, semantic matching is more natural. For the examples in this paper, we use keyword matching only, however, the method can be extended to semantic matching [12] as well.

3.2 Process and Interface Oriented Specifications

From the viewpoint of web services' users, specifications of required composite web services can be separated into two categories:

- Process Oriented Specification (POS) contains a set of abstract component web services in addition to the business logic describing the relationship among these services;
- Interface Oriented Specification (IOS) contains a set of input and output data (possibly with pre-conditions) which specifies a user's request for web service. In this case no information about a structure or possible components of the requested service is available.

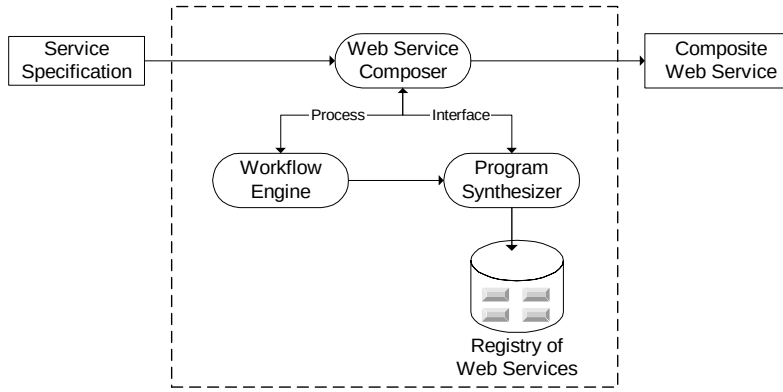

Fig. 1. The Infrastructure of Web Services Composition

Using POS the user provides desired abstract web services (by functionality) as well as a business logic which describes the relationship among the services. Using IOS the user specifies only desired input/output of a required web service. The internal structure of this web service can be hidden from the user.

The above-mentioned categories of service specifications may require different methods and tools. In particular, we think that POS of composite web service is better modeled by a workflow which is generated by business model, while the IOS of service can be better implemented via usage of an automatic program synthesis method.

A possible infrastructure for web services composition is presented on Figure 1.

In this infrastructure, if the service specification is process oriented then it is processed by the workflow engine which generates a workflow model for the composite service as a solution. The workflow model can be described by a workflow language or web service flow language (e.g. WSFL [9]). If the service specification is interface oriented (by arguments and results of the service) then the requirement is transferred to the Program Synthesizer. If there is a service specification (in the registry of available services) which matches the specified requirements then the service is selected. Otherwise, the Program Synthesizer starts a composition process using available service specifications and a program synthesis method.

In this paper we are focused on the Program Synthesizer component of the above infrastructure. For research on building the Workflow component of the infrastructure we refer, for example, to [5].

4 Service Composition Method

We assume that the functionality of available web services can be presented as input-output specifications defining what is a source for the service and what

will be a result of the service performing. In this case a web service request can also be expressed as an input-output specification.

We use a Structural Synthesis Program (SSP) method [11] for building a composition of services. We choose the SSP method because of its efficiency in basic cases, well-developed formal foundations, ability to extract an action sequence from the process of services composition and our previous experience in usage and implementation of the method. However, we would like to underline that SSP is not the only method that could be used for our problem solving. Any other synthesis method which allows combination of input-output specifications and extraction of action sequence can also be applied.

Let us consider a value-added service provision problem from a service composition perspective. The web service request for the ski buying problem can be described as an input-output specification where the input contains customer's body height in inches and the requested output is ski prices in NOK (here NOK stands for "Norwegian Krone").

From the service providers' side we assume that there are the following specifications of services:

- the core service taking ski length in inches as input and providing ski prices in USD for available skies as the output
- the value-added service taking ski prices in USD as input and providing ski prices in NOK (or in another local currency) as output
- the value-added service taking measurements in inches as input and providing measurements in centimeters (cm) as output
- the value-added service taking measurements in centimeters as input and providing measurements in inches as output
- the value-added service taking body height measurements in centimeters and providing a recommended ski length in centimeters as output

As we can see, there is no single service that matches the problem of buying skies based on body measurements in inches and prices in NOK. However, services combination can provide the required service.

As we mentioned above, we use SSP for service composition. In SSP, component specifications are presented as implications of the following two types (where underlining means conjunction of the underlined components):

- unconditional computability statements

$$\underline{A} \underset{f}{\rightarrow} B \tag{1}$$

  where f is a term representing the function which computes the realization b of B from the realization a of A.
- conditional computability statements

$$\underline{(\underline{A} \underset{g}{\rightarrow} B)} \rightarrow (\underline{C} \underset{F}{\rightarrow} D) \tag{2}$$

  where g is a variable representing a function which must be synthesized in order to compute b from $\underline{a}$ and F is a term representing computation of d depending on $\underline{g}$

Using SSP specification the above-mentioned single services can be expressed as follows:

- the core service

$$ski\_lenght\_in\_inches \xrightarrow[ski\_prices\_service]{} ski\_prices\_USD \tag{3}$$

- the measurements converter service:

$$body\_height\_in\_inches \xrightarrow[inch\_cm\_converter]{} body\_height\_in\_cm \tag{4}$$

$$ski\_length\_in\_cm \xrightarrow[cm\_inch\_converter]{} ski\_lenght\_in\_inches \tag{5}$$

- the currency converter service:

$$ski\_prices\_USD \xrightarrow[USD\_NOK\_converter]{} ski\_prices\_NOK \tag{6}$$

- recommendation service:

$$body\_heigth\_in\_cm \xrightarrow[ski\_recommendation\_cm]{} ski\_lenght\_in\_cm \tag{7}$$

We would like to notice that there can be more than one service provider for each service. Their specifications then will differ only in the name of service provider under the arrow.

The request for service (or goal) can be expressed as follows:

$$body\_heigth\_in\_inches \rightarrow ski\_prices\_NOK \tag{8}$$

The SSP-based synthesizer takes a goal (8) and makes a plan for its fulfillment using (4), (7), (5), (3) and (6). The composed service is built by applying the SSP inference rules described in [11]. We also would like to underline that the result of planning with SSP is a sequence of actions (services) to be performed. In particular, informal meaning of the synthesized plan (composed services) can be as follows:

1. use the *inch_cm_converter* service for converting body height measurements
2. use *ski_recommendation_cm* service for getting recommended ski length in cm
3. use *cm_inch_converter* service for converting ski length measurements
4. use the *ski_prices_service* core service
5. use *USD_NOK_converter* service

The above-mentioned sequence is a synthesized composed service.

We would like to notice that usage of single names in computability statements might lead to a big overhead in service description. In particular, we may need to describe all converters for measurements and currency for all components (for body height, for ski length etc). This problem is solved in SSP on a specification level. It is allowed usage of compound names there. For example, it is possible to specify the `Measurements` and `Currency` classes as follows:

```
Measurements
    inches, cm : numeric;
    inches ——————————→ cm
              inch_cm_converter
    cm ——————————→ inches
         cm_inch_converter

Currency
    USD, NOK : numeric;
    USD ——————————→ NOK
         USD_NOK_converter
    NOK ——————————→ USD
         NOK_USD_converter
```

Usage of these classes can be as follows:

```
body_height, ski_length : Measurements;
ski_prices : Currency;
```

This allows usage of compound names like `body_height.cm`, `body_height.inches` which refer to the above specified cm-to-inches and inches-to-cm services without necessity to define such services explicitly for all possible arguments.

Usage of full SSP power in our simplified working example seems to be too advanced, however, in case of tens or hundreds of specified services it gives big advantage.

In a more complex case the synthesizer may generate more than one possible service composition. This may lead to alternative possible ways of the service provision and to necessity of applying fine-grain selection (either automatically or manually) for a particular service selection.

We also would like to notice that using conditional computability statements in service presentation allows us synthesize conditional composed services. In particular, alternative conditional services or exceptions can be described by different left-hand-side implications in computability statements of the form (2) and a selection criterion as a right-hand-side implication of the form (2). Such conditional computability statements can be generated automatically from unconditional computability statements (1) extended by disjunction on the right-had-side of the implication [10].

5 Service Description Languages

In spite of nice features of the SSP-based language and synthesis we don't think that this language should be considered as a general service description language (it is rather suitable as an internal language for synthesis). Our approach is to use a widely accepted language for service description and to provide a translation from this language to SSP presentation when it is needed.

For our purpose (no matter what description language is used) the content of the web service description should allow:

- to specify service inputs and outputs. This can be used by matchmaker to determine whether the service meets request;

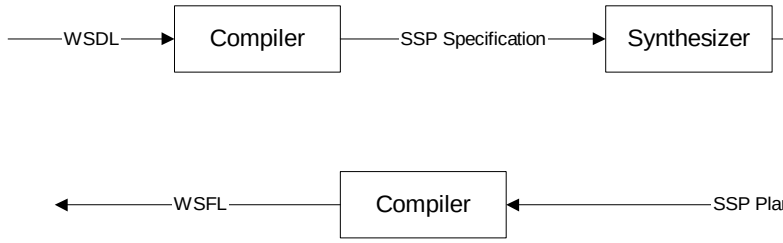

Fig. 2. Compilers and Synthesizer

- to describe a service as a task which should be performed. The task should be described in some form which allows its invocation;
- to specify the details of how the web service can be accessed. Typically a protocol will specify a communication protocol (eg., RPC, HTTP-FORM, CORBA IDL, SOAP, Java RMI/Bean) and the location (eg., URL, port number)

Recent industrial efforts have developed several proposals for improving web services discovery and execution. Most significant initiatives include the Universal Description, Discovery, and Integration (UDDI) [15]; the XML-based Web Service Description Language (WSDL) [17]; and DAML-S, a DAML (DARPA Agent Modelling Language) based semantic markup of web services [6].

We don't want to restrict ourselves by only one description language or one internal language and allow different languages to be used in our system. In order to support that, we have to design a set of compilers for translation between the external XML based web services language and the internal logic language. Figure 2 shows the dataflow between compilers and synthesizer. Here, WSDL is a general term for all web services description languages (in particular, this can refer to DAML-S profile). WSFL also refers to any web services flow language, for example DAML-S process model.

In particular, this means that synthesizer can be developed independently of particular internal representation (assuming that whatever representation is used an access to it is implemented via standard methods). Such approach requires development of different compilers from external description languages to different internal representations.

In our prototype, we choose DAML-S as external language (however, other external languages can be added in a similar way). A basic reason for this choice is that DAML-S has features for support of both web services selection and composition. For more detail specification and examples about DAML-S we refer to [6].

Taking SSP-based language as internal representation and DAML-S as external language we have implemented a DAML-S compiler which translates between the DAML-S documents and SSP specifications. For example, a description of the core service in our working example can be as follows[1]:

[1] &concepts is the shorthands for the URI of the concepts' name space

```
<service:ServiceProfile rdf:ID="Ski_Price_Service">
 <input>
  <profile:ParameterDescription rdf:ID="ski_length">
   <profile:parameterName>ski_length</profile:parameterName>
   <profile:restrictedTo rdf:resource="&concepts;#Inch" />
   <profile:refersTo rdf:resource="&concepts;#Measurements" />
  </profile:ParameterDescription>
 </input>
 <output>
  <profile:ParameterDescription rdf:ID="ski_price">
   <profile:parameterName>ski_price</profile:parameterName>
   <profile:restrictedTo rdf:resource="&concepts;#USD" />
   <profile:refersTo rdf:resource="&concepts;#Currency" />
  </profile:ParameterDescription>
 </output>
</service:ServiceProfile>
```

This description can be translated into the following SSP formula:

$$ski\_length\_in\_inch \xrightarrow[ski\_price\_service]{} ski\_prices\_USD$$

We would like to notice that a convention of interface names is needed to make sure that the interface parameters which refer to the same concept have the same names. In some web services language, such as DAML-S, RDF resources and domain restrictions are used to indicate the concept and range of a parameter name. The compiler takes care about the problem and it can tranlate all restrictions to SSP specifications.

6 Conclusion

We propose an approach to automate value-added web services composition. The problem of service composition is considered as a problem of software synthesis where algorithms for matching and composition are based on the SSP method. The SSP language was adopted as internal presentation language for automated service composition, while DAML-S is used as external language for description of web service properties.

Basic features of the proposed approach can be formulated as follows:

- Service composition is based on the input-output information of services components and requires little domain knowledge.
- Usage of compilers allows adopting different internal formal languages and external XML-based web service description languages.

At the moment, the basic components of the approach are developed. We currently work on developing of presentation tools for making more convenient usage of the system by the customer.

The approach can work together with other workflow methods. We also suggest that combining workflow model and synthesis method may give a better efficiency and flexibility in web service provision. We consider this direction as an interesting future work. Another our future work is related to taking into consideration semantic matching both for service composition and optimization of the results of the composition.

Acknowledgements

This work is partially supported by the Norwegian Research Foundation in the framework of the Information and Communication Technology (IKT-2010) program - the ADIS project and in the framework of the Distributed Information Technology (DITS) program - the ElComAg project.

References

1. Anupriya Ankolekar et al. Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop*, 2001.
2. B. Benatallah, M. Dumas, M.C Fauvet, and F. Rabhi. Towards patterns of web services composition. Technical report, The University of New South Wales, November 2001.
3. James L. Caldwell. Moving proofs-as-programs into practice. In *The twelfth IEEE International Automated Software Engineering Conference*, 1997.
4. Fabio Casati, Mehmet Sayal, and Ming-Chien Shan. Developing e-services for composing e-services. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceeding of 13th Int. Conference on Advanced Information Systems Engineering (CAiSE), Interlaken, Switzerland*. Springer Verlag, June 2001.
5. F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and dynamic service composition in eflow. In Benkt Wangler and Lars Bergman, editors, *Proceeding of 12th Int. Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden*. Springer Verlag, June 2000.
6. DAML-S Coalition. Daml-s 0.6 draft release. http://www.daml.org/services/daml-s/2001/10/, december 2001.
7. R. Doorenbos, O. Etzioni, and D. Weld. A scalable comparison-shopping agent for the world wide web. In *Proceedings of the First International Conference on Autonomous Agent(Agent '97)*, Marina Del Rey, CA, February 1997.
8. R. Guttman, A. Moukas, and P. Maes. Agent-mediated electronic commerce: A survey. *Knowledge Engineering Review*, 1998.
9. IBM. *Web Services Flow Language(WSFL 1.0)*.
10. Sven Lammermann. *Runtime Service Composition via Logic-Based Program Synthesis*. PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, Stockholm, 2002.
11. Mihhail Matskin and Enn Tyugu. Structural synthesis of programs and its extensions. *Computing and Informatics Journal*, 20:1–25, 2001.
12. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of web services capabilities. In *First International Semantic Web Conference (ISWC)*, Sardinia, Italy, June 2002.

13. John J. Penix. *Automated Component Retrieval and Adaptation Using Formal Specifications.* PhD thesis, Division of Research and Advanced Studies of the University of Cincinnati, 1998.
14. H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference processbased multi-enterprise processes. In Benkt Wangler and Lars Bergman, editors, *Proceeding of 12th Int. Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden.* Springer Verlag, June 2000.
15. UDDI.org. Universal description, discovery and integration specification. http://www.uddi.org/specification.html.
16. T. Uustalu, U. Kopra, V. Kotkas, M. Matskin, and E. Tyugu. The nut language report. Technical report, The Royal Institute of Technology(KTH), 1994.
17. W3C.org. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl.