Application of Linear Logic to Web Service Composition

Jinghai Rao, Peep Küngas
Department of Computer and Information Science,
Norwegian University of Science and Technology,
N-7491, Trondheim, Norway

Mihhail Matskin
Department of Microelectronics and Information Technology,
Royal Institute of Technology,
SE-164 40 Kista, Sweden

Abstract ***In this paper, we propose a method for automated web service composition by applying Linear Logic (LL) theorem proving. We distinguish value-added web services and core service by assuming that the core service is already selected by the user, but its functionality does not completely match the user's requirement. Our method enables automation for combining the core service together with a set of value-added services to solve the problem. The method uses web service languages for external presentation of atomic web services (WSDL) or composite web services (BPEL4WS), while the services are internally presented by extralogical axioms and proofs in LL. In this paper, we are focused on the internal presentation and proof. LL, as the internal representation language, enables us to define some issues required by web service composition formally, such as qualitative and quantitative constraints plus subsumption reasoning on concepts. In addition, LL guarantees the correctness and completeness of service composition process.***

Keywords: Web Service Composition, Linear Logic

1 Introduction

Web services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the web [1]. Several initiatives with web services provide platforms and languages that should allow easy integration of heterogeneous systems. In particular, such Web languages as Universal Description, Discovery, and Integration (UDDI) [2], Web Services Description Language (WSDL) [3], Simple Object Access Protocol (SOAP) [4] and part of DAML-S [5] ontology, such as service profile and service grounding, define standard ways for service discovery, description and invocation (message passing). Some other initiatives such as Business Process Execution Language for Web Service (BPEL4WS) [6] and DAML-S Process Model, are focused on representing service compositions where flow of a process and bindings between services are known a priori.

The ability to efficiently select and integrate inter-organizational and heterogeneous services on the web at runtime is a critical step towards the development of the online economy. In particular, if no single web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfill the request. One challenge is that web services can be created and updated on the fly and it may be beyond human capabilities to analyze the required services and compose them manually. Another challenge is that web services can be developed by different organizations, which use different semantic models to describe the services. As a result, the ability to efficient integration of possibly heterogeneous services on the Web becomes a complex problem (especially for dynamic composition during runtime).

On the other hand, if we consider atomic services as software components, the web service composition can be presented as a software synthesis problem. In comparison with

software components, web services may present a higher abstraction level and they are more loosely coupled. This may allow us to construct a composite service based on a functional specification, without taking into account low level technical details, such as operating system, communication protocol or programming language. Taking this into account, advantages of using automatic software synthesis can be described as follows:

- automatic software synthesis provides an ability to construct composite services in dynamically changing environments, especially when there is a huge number of services available;
- the resulting composite services (generated automatically using sound and complete rules) are correct wrt specification of atomic services.

In this paper, we describe a method for automated web service composition which is based on the proof search in (propositional) multiplicative intuitionistic fragment of Linear Logic (MILL [7]). Given a set of existing web services and a set of functionality and non-functionality attributes, the method finds a composition of atomic services that satisfies the user requirements. The composition process in our case is shown in Fig. 1. First, a description of existing web services (written in WSDL) is translated into extralogical axioms in Linear Logic (LL) [8], and the requirements to the composite services are specified in form of a LL sequent to be proven. Second, we use a MILL theorem prover to determine whether the requirements can be fulfilled by the composition of existing atomic service. If it is possible then the last step is to construct flow models (written in BPEL4WS) from the generated proofs. We assume that the composite service is ready to be executed when the flow model and description of each atomic services are given. Because of soundness of MILL correctness of composite services is guaranteed with respect to initial specification. Completeness of MILL ensures that all composable solutions would be found. Our method can be regarded as an extension of the service composition method using Structural Synthesis of Programs proposed in [9]. The difference is that the method used in this

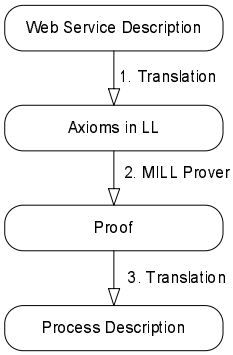

Figure 1: The process of web service composition

paper considers also the quantitative and qualitative constraints in addition to the structural information of services.

This paper is focusing on the second step of the composition process, namely presentation and proof using LL. We assume that WSDL presentation of services has been translated into a set of LL axioms by a compiler (we will explain this translation in details in a separate publication).

The complete automation of service composition is still a very complex problem and it requires further research. In this paper we restrict the problem to composition of value-added services assuming that the core service (atomic or composed) already exists. The rest of this paper is organized as follows: Section 2 presents a motivating example, Section 3 describes LL and its usage in service composition, Section 4 discusses the usage of type system to enable semantic composition, and the last section concludes the paper.

2 Motivating Example

Value-added services differ from core services—they are not a part of core services but act as complements to the core services. 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 combination and they may provide incremental extension of core services. For example, in online shopping, the core services are product search, ordering, payment and shipment. However, some other services, such as currency exchange,

measurement converter, language translation can also be required in a case when the core services cannot meet the users' requirements. Those services are not designed for a particular core service but they rather extend an ability of core services and add value to the core services.

As a working example we consider a ski selling web service. A core service, in this case, receives characteristics of a pair of skis (length, brand, model etc) and provides prices, availability and other requested characteristics as output. We assume that user would like to use this service but there are gaps between the user's requirements and the functionalities the service provides.

The differences could exist, for example, in the following details:

- the user would like to receive local currency (such as Norwegian Krone) as price but the service provides price in US Dollars only;
- the user would like to use centimeters as length measurement units but the service uses inches;
- the user does not know what ski length or model is needed and s/he would like that they can be calculated from his/her height and weight;
- the user doesn't know which brand is most suitable and s/he would like to get a recommendation from a web service.

Here we illustrate a case that considers only the functionality attributes, basically input and output of a service. However, our method is able to consider the non-functionality attributes as well.

We assume the user provides the body height measured in centimeters (cm), the body weight measured in kilograms (kg), his/her skill level and the price limit. The user would like to get a price of recommended pair of skis in Norwegian Krone (NOK).

The core service *selectSkis* accepts the ski length measured in inch, ski brand, ski model and gives the ski price in US Dollars (USD).

The available value-added services are as follows :

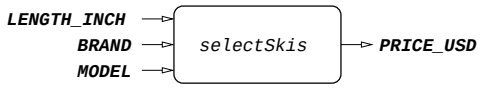

Figure 2: The core service for buying skis.

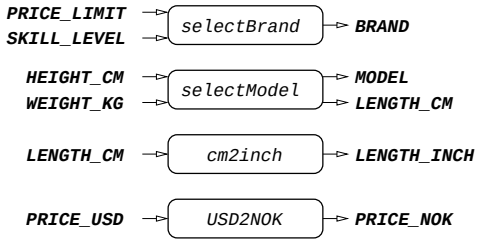

Figure 3: Available value-added services.

- *selectBrand*—given a price limit and a skill level, provides a brand;
- *selectModel*—given body height in cm and body weight in kg, provides ski length in cm and a model;
- *cm2inch*—given ski length in cm provides ski length in inches;
- *USD2NOK*—given ski price in USD provides ski price in NOK.

The core service and available value-added services are depicted respectively in Fig. 2 and Fig. 3. A required service is presented in Fig. 4. The structure of a solution for the required service is represented in Fig. 5.

We would like to mention that our working example is made simpler than required for practical cases. This has been done in order to keep simplicity of presentation. In more practical cases there can be more value-added services available and more parameters for the core service. Therefore, the user would not have an ability to find a solution intuitively. In addition, it is also beyond the user's ability to search the huge amount of available value-added services to find all possible solutions. In particular, if the set of possible solutions consists of all existing converters to all inputs and

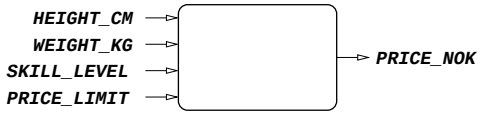

Figure 4: The required service for buying skis.

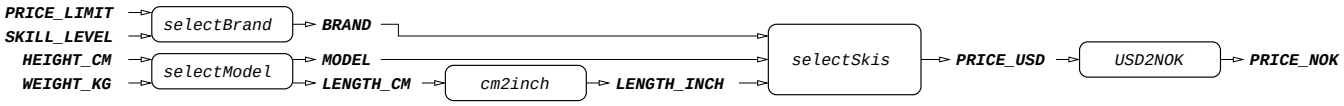

Figure 5: The final service structure for buying skis.

outputs of all web services (both core and value added), this may cause big overhead in service provision. Taking this into account, we think that automatic composition would be an efficient and practical solution in this case.

3 Service Composition Method

3.1 Functionality and Non-functionality Attributes

When we present the web service specification, we distinguish functionality attributes and non-functionality attributes of a single service. In [10], the authors give the definitions of functionality and non-functionality attributes that are used to specify the service profile in DAML-S. Service functionalities are represented as a transformation from the inputs required by the service to the outputs produced by the service. Non-functionality attributes are other properties than functionalities that can be used to describe a service, for example, price, location, quality of the service.

The functionality attributes are used in composition for connecting atomic services by means of inputs and outputs. The composition is only possible if output of one service could be transferred to another service as input.

Non-functionality attributes are useful in evaluating and selecting service when there are many services that have the same functionality attributes. In the user's requirement, the non-functionality attributes are specified as constraints. In general, those constraints can be put into three categories:

- **quantitative constraints for composite service:** These constraints limit the resources consumed by the composite service. The total resource is the sum of all atomic services that form the composite service. For example, the price of composite service is the sum of prices for all included atomic services.
- **quantitative constraints for atomic service:** These constraints limit the resources consumed by each single atomic service. For example, the user can specify the price limit for each included service.
- **qualitative constraints:** Some attributes, such as service type, service provider or geographical location, can't be expressed by quantities. We call them qualitative attributes. The constraints to qualitative attributes propagate to all included services. For example, if the user requires that the composite service locates in Norway, all included service should also reside in Norway.

In the next section, we will illustrate presentation of quantitative constraints (e.g. cost of executing web service) and qualitative constraints (e.g. location constraints) by LL. Those constraints are considered during proof search. In other words, if an atomic service does not meet the constraints, it cannot be selected.

3.2 Linear Logic for Web Service Composition

LL is a refinement of classical logic introduced by J.-Y. Girard [8] to provide a means for keeping track of "resources"—in LL two assumptions of a propositional constant A are distinguished from a single assumption of A. Although LL is not the first attempt to develop resource-oriented logics (well-known examples are relevance logic and Lambek calculus), it is by now one of the most investigated ones. Since its introduction LL has enjoyed increasing attention both from researchers in proof theory and computer science. Therefore, because of its maturity and well-developed semantic, LL is useful as a declarative language and inference system.

MILL fragment of LL consists of multiplicative conjunction $\otimes$, linear implication $\multimap$ and "of course" (!) operator. In terms of resource

acquisition the logical expression $A \otimes B \multimap C \otimes D$ means that goals C and D are obtainable only if both resources A and B are available. Thus the connective $\otimes$ defines deterministic relations between resources. After literals A and B are consumed, literals C and D are generated. In that way we can encode different behaviors of computations. The formula $!A$ means that we can use or generate a literal A as much as we want—the amount of the resource is unbounded. While in classical logic literals may be copied by default, in LL it has to be stated explicitly.

Generally, the requirements for a composite service could be expressed by a logical formula:

$$(\Gamma_v, \Gamma_c); \Delta \vdash I \multimap O$$

where both Γ_v and Γ_c are sets of extralogical axioms representing available value-added web services and core services respectively, Δ is a conjunction of non-functionality constraints. The constraints could be used either for specifying quantitative or qualitative attributes of required services. $I \multimap O$ is a functionality description of the required composite service. Both I and O are conjunctions of literals, I represents the set of input parameters of the service and O represents output parameters produced by the service. Intuitively, the formula can be explained as follows: given a set of available atomic services and the constraints, try to find a combination of services that computes O from I. Every element in Γ_v and Γ_c is in form $\Delta \vdash I \multimap O$, whereas meanings of Δ, I and O are the same as described above.

Here, we illustrate the LL presentation of our motivating example taking both functionalities and non-functionalities into consideration.

The available value-added services are specified as follows:

$$\Gamma_v = \begin{array}{l} NOK^{10} \vdash PRICE\_LIMIT \otimes SKILL\_LEVEL \multimap_{selectBrand} BRAND \\ \vdash HEIGHT\_CM \otimes WEIGHT\_KG \multimap_{selectModel} LENGTH\_CM \otimes MODEL \\ NOK^{20} \vdash LENGTH\_CM \multimap_{cm2inch} LENGTH\_IN \\ LOC\_NORWAY \vdash PRICE\_USD \multimap_{USD2NOK} PRICE\_NOK \end{array}$$

NOK^{10} in the left hand side of recommendation service *selectBrand* denotes $\underbrace{NOK \otimes \ldots \otimes NOK}_{10}$. This means that 10 NOK are consumed by executing the service. The *cm2inch* service costs 20 NOK and other services are for free. In the specification it is also said that the currency converting service *USD2NOK* is performed within Norway and for other services it does not matter where they are performed.

The core service is specified by:

$$\Gamma_c = \vdash LENGTH\_IN \otimes BRAND \otimes MODEL \multimap PRICE\_USD$$

The constraints for the composite service are as follows:

$$\Delta = NOK^{40} \otimes !LOC\_NORWAY$$

This means that we would like to spend 40 NOK for the composite service and all location-aware services have to be performed within Norway ($!LOC\_NORWAY$). ! symbol in $!LOC\_NORWAY$ describes that we allow unbound number of services from Norway in the composite service. Here, we consider quantitative constraints (price) as regular resources in LL. If the total number of resources required by services, determined by functionality attributes, is less than the number of available resources, the services can be included into composite service. Otherwise, for example, if the *selectBrand* service would require 30 NOK for execution, the total required amount would be 50 NOK and the composition is not valid.

For the qualitative constraints (location), the service uses a literal LOC_NORWAY to determine its value and we can determine in the set of requirements Δ whether a service meets the requirement. However, if there is no such literal in the service description, the constraint does not apply to that service at all.

Finally, the requirements for the composite service are specified as follows:

$$(\Gamma_v, \Gamma_c); \Delta \vdash HEIGHT\_CM \otimes WEIGHT\_KG \otimes PRICE\_LIMIT \otimes SKILL\_LEVEL \multimap PRICE\_NOK$$

The required service can be proven to be correct (and then extracted from the proof) from the specification of available value-added services and the core service. Due to the lack of space we do not represent the complete proof. However a small fragment in Fig. 6 shows clearly how dependencies between services are discovered and the resulting composite service is constructed by the LL theorem prover. Additionally all details considering

non-functionality constraints have been omitted here for reducing size of the proof fragment. The inference rules of MILL and the complete proof of a similar problem can be found in [11].

4 Composition using Semantic Description

To make service composition more flexible, we should consider a case where two services can be connected together, even if the output parameters of one service does not match exactly the input parameters of another service. In general, if a type assigned to the output parameter for service A is a subtype of the type assigned to an input parameter for service B, it is safe to transfer data from the output to the input. In addition, if we consider resources and goals in LL, the subtyping is used in two cases: 1) given a goal x of type T, it is safe to replace x by another goal y of type S, as long as it holds that T is a subtype of S; 2) conversely, given a resource x of type S, it is safe to replace x by another resource y of type T, as long as it holds that T is a subtype of S.

Such subtyping rules can be applied to either functionality (parameters) or non-functionality constraints. Here we use two examples to illustrate the basic idea. First, we show a simple case how two services with different types of parameters are composed. Let us assume that the output of *selectModel* service is SKI_LENGTH_CM, while the input of *cm2inch* service is $LENGTH\_CM$ that is more general. So it is safe to transfer the more specific output to the more general input. Another example considers the qualitative constraints. If we require a service which is performed within Europe, we regard Europe as a resource. Because Norway is a country in Europe, it is safe to replace Europe with Norway. Intuitively, if we require a service that is performed within Europe, the service performed within Norway meets such requirement.

In this paper, we assume that the service requester and the service provider use a shared ontology to denote concepts in description of services. Otherwise, the ontology interoperability is a big issue that is beyond a scope of this paper. WSDL has basic capabilities to define simple relationships between data types and it uses XSD as the canonical type system. The user is able to define the subsumption on taxonomies of data types as it is usually done for XML schema. In this case, the subtyping rules mentioned above could be applied to the services that are described by WSDL. However, WSDL is not designed as an ontology language. If we require richer semantic information and more complex relations, we have to link the WSDL types to other ontologies that describe particular object types and their features, for example, the ontology written in OWL or DAML+OIL. [12] proposes a prototype to combine the DAML-S semantic service description with invocation information of the WSDL descriptions.

5 Conclusion and Future Work

In this paper, we propose a method for value-added web service composition. We distinguish value-added web services and core service by assuming that the core service is already selected by the user, but its functionality does not completely match the user's requirement. Our method enables automation for finding a composition of the core service together with a set of value-added services to solve the problem.

The method uses web service languages for external presentation of atomic web services (e.g. WSDL) or composite web services (e.g. BPEL4WS), while the services are internally presented by extralogical axioms and proofs in LL. We are focused on the internal presentation and proof in this paper. Using LL for internal representation language we can formally define some issues required by web service composition, such as qualitative and quantitative constraints and subsumption reasoning on concepts. In addition, LL guarantees the correctness and completeness of composite services. We have also implemented a theorem prover for first-order MILL and performed initial experiments. The theorem prover is available at `http://www.idi.ntnu.no/~peep/RAPS`.

Despite the advantages of LL over other formal frameworks, LL has been barely considered for either web services composition or software synthesis. This may be due to its higher computational complexity compare to other logics. However, our first experiments

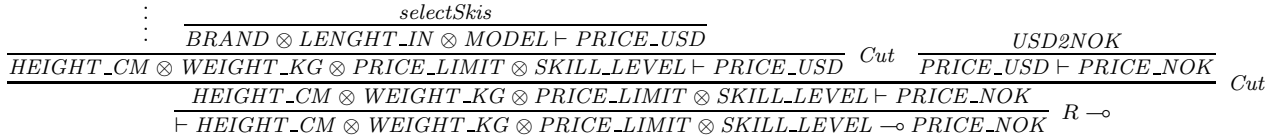

Figure 6: A fragment of the LL proof for composing the ski buying service.

show that efficient theorem proving is available at least for MILL fragment of LL, if certain proof search strategies are considered.

The composition of core services usually needs business model to specify the relationship between multiple core services. The method presented in this paper is also valid for core service composition if the business model is given.

Our current work is directed to integrating all steps of the web service composition process presented in Fig. 1. In other words, we are working on integration of the theorem prover with the compilers for the semantic description of web services and the LL axioms.

We are also working on adding the disjunction connective to the logical specification of service output. This is useful when we should consider exceptions or optional outputs of atomic services.

Acknowledgment

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.

References

[1] Ibm web services tutorial.

[2] Tom Bellwood et al. Universal description, discovery and integration specification (uddi) 3.0.

[3] Erik Christensen et al. Web services description language (wsdl) 1.1.

[4] Don Box et al. Simple object access protocol (soap) 1.1, 2001.

[5] David Martin et al. Daml-s 0.7 draft release, January 2003.

[6] F. Curbera et al. Business process execution language for web services (bpel4ws) 1.0, July 2002.

[7] Patrick Lincoln. Deciding provability of linear logic formulas. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *London Mathematical Society Lecture Note Series*, volume 222, pages 109–122. Cambridge University Press, 1995.

[8] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[9] Mihhail Matskin and Jinghai Rao. Using structual synthesis of programs to facilitate web services composition. In C. Bussler et al., editors, *The first workshop of Web Services, e-Business and the Semantic Web, Toronto, Canada*, volume LNCS 2512. Springer Verlag, May 2002.

[10] Anupriya Ankolekar et al. Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop*, 2001.

[11] Peep Küngas and Jinghai Rao. Linear logic for structural software synthesis. submitted.

[12] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure" workshop in conjunction with ICEIS2003*, 2003.