Toward the Composition of Semantic Web Services

Jinghai Rao and Xiaomeng Su

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

Abstract. This paper introduces a method for automatic composition of semantic web services using linear logic theorem proving. The method uses semantic web service language (DAML-S) for external presentation of web services, and, internally, the services are presented by extralogical axioms and proofs in linear logic. Linear logic(LL)[2], as a resource conscious logic, enables us to define the attributes of web services formally (in particular, qualitative and quantitative value of non-functional attributes). The subtyping rules that are used for semantic reasoning are presented as linear logic implication. We propose a system architecture where the DAML-S parser, linear logic theorem prover and semantic reasoner can work together. This architecture has been implemented in Java programming language.

1 Introduction

The Grid is a promising computing platform that integrates resources from different organizations in a shared, coordinated and collaborative manner to solve large-scale science and engineering problems. The current development of the Grid has adapted to a services oriented architecture and, as a result, recently Grid technologies are evolving towards an Open Grid Services Architecture (OGSA).The convergence of Web services with Grid computing will accelerate the adoption of Grid technologies. [1] defines a Grid service as a Web service that provides a set of well-defined interfaces and follows specific conventions. As such, Grid service will inherently share some of the same problems and technical challenges of Web service in general.

The ability to efficiently and effectively select and integrate inter-organizational 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 rapidly. However, the task of web service composition is a complex one. Firstly, 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. Secondly, the web services are developed by different organizations that use different semantic model to describe the features of services. The different semantic models indicate that the matching and composition of web service have to take into account on the semantic information.

In this paper, we propose a candidate solution which we believe to contribute to solving the two challenges. 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 [3]). The idea is, given a set of existing web services and a set of functionality and non-functional attributes, the method finds a composition of atomic services that satisfies the user requirements. The fact that Linear Logic is resource conscious makes it possible to make proving on both qualitative and quantitative non-functional attributes of web services. Because of the soundness of the logic fragment correctness of composite services is guaranteed with respect to initial specification. Further, the completeness of logic ensures that all composable solutions would be found.

The rest of this paper is organized as follows: Section 2 presents a system architecture for composition of semantic web services. Section 3 presents the methods on transformation between DAML-S documents and Linear Logic axioms. Section 4 discusses the usage of type system to enable semantic composition. Section 5 is the related works and the conclusion of the paper.

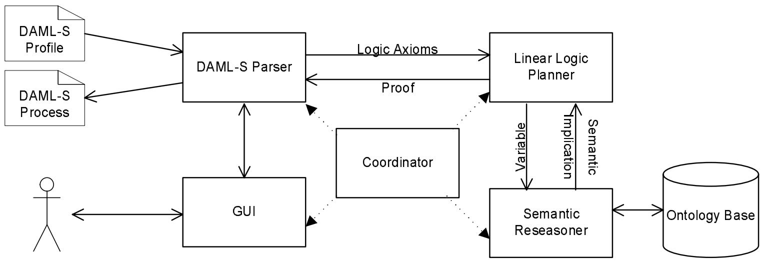


Fig. 1. Architecture for Service Composition

2 The Service Composition Architecture

Figure 1 depicts the general architecture of the proposed web service composition process. The approach is presented by the following process. First, a description of existing web services(in the form of DAML-S Profile) is translated into axioms of Linear Logic, and the requirements to the composite services are specified in form of a Linear Logic sequent to be proven. Second, the Linear Logic Theorem Prover determines whether the requirements can be fulfilled by composition of existing atomic services. On reading each propositional variable, the theorem prover requires the semantic reasoner to provide possible subtyping inference. The subtypings are inserted into the theorem prover as logic implications. If one or more proofs are generated the last step is the construction of flow models (written in DAML-S Process). The process is controlled by the coordinator, especially when components are distributely located. During the process, the user is able to interact with the system by GUI.

In this paper, we pay special attention on the DAML-S Parser and the Semantic Reasoner. The detail on theorem proving part has been already introduced in [7]. The readers who have knowledge about Linear Logic or theorem proving are able to understand this part easily without referring to the separate publication.

3 Transforming from DAML-S to Linear Logic Axioms

In our system, the web services are specified by DAML-S profile externally and presented by LL Axioms internally. The transformation is made automatically by DAML-S Parser. The detail presentation of DAML-S can be found in [4]. Here, we focus on the presentation of LL axioms.

Generally, a requirement to composite web service, including functionalities and non-functional attributes, can be expressed by the following formula in LL:

$$\Gamma; \Delta_1 \vdash (I \multimap O) \otimes \Delta_2$$

where Γ is a set of logical axioms representing available atomic web services, Δ_1 is a conjunction of non-functional constraints. Δ_2 is a conjunction of non-functional results. We will distinguish these two concepts later. $I \multimap O$ 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 non-functional attributes, try to find a combination of services that computes O from I. Every element in Γ is in form $\Delta_1 \vdash (I \multimap O) \otimes \Delta_2$, whereas meanings of Δ_1, Δ_2, I and O are the same as described above.

Next, we describe the detail procedure to transform a DAML-S document into the linear logic expression. We present in sequence the transformation on functionalities and non-functional attributes. Afterwards, we present the whole process by an example.

3.1 Transforming on Functionalities

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

The web service is presented by DAML-S profile externally. The functionality attributes of the "ServiceProfile" specifies the computational aspect of the service, denoting by the input, output, precondition and postcondition.

Below is an example of the functionalities for a temperature report service:

```
<profileHierarchy:Information_Service rdf:ID="Temperature">
  <service:presentedBy rdf:resource="&service;#Temperature"/>
  <profile:serviceName>Temperature_Report</>
  <profile:textDescription>
      The service provides the temperature of the city with the given zip code
```

```
    </profile:textDescription>
    <profile:input>
      <profile:ParameterDescription rdf:ID="ZipCode">
        <profile:parameterName>ZipCode</profile:parameterName>
        <profile:restrictedTo rdf:resource="&zo;#ZipCode"/>
        <profile:refersTo rdf:resource="&model;#ZipCode"/>
      </profile:ParameterDescription>
    </profile:input>
    <profile:output>
      <profile:ParameterDescription rdf:ID="CelsiusTemp">
        <profile:parameterName>CelsiusTemp</profile:parameterName>
        <profile:restrictedTo rdf:resource="&wo;#CelsTemp"/>
        <profile:refersTo rdf:resource="&model;#CelsiusTemp"/>
      </profile:ParameterDescription>
    </profile:output>
  </profileHierarchy:Information_Service>
```

From the computation point of view, this service requires an input that has type "$\&zo;\#ZipCode$" and produces an output that has type "$\&zo;\#CelsTemp$", the value of temperature measured in Celsius. Here, we use entity types as a shorthand for URIes. For example, $\&zo;\#ZipCode$ refers to the URI of the definitions for zip code parameter: `http://www.daml.org/2001/10/html/zipcode-ont\#ZipCode` When translating to Linear Logic formula, we translate the field "restrictedTo" (variable type) instead of the parameter name, because we regard the parameters' type as their specification. Below is the example propositional linear logic formula that expresses the above DAML-S document:

$$\vdash \&zo;\#ZipCode \multimap_{\&service;\#Temperature} \&tc;\#CelsTemp$$

3.2 Non-functional Attributes

Non-functional attributes are useful in evaluating and selecting service when there are many services that have the same functionalities. In the service presentation, the non-functional attributes are specified as facts and constraints. We classify the attributes into four categories:

- **Consumable Quantitavie Attributes:** These attributes limit the amount of resources that can be consumed by the composite service. The total amount of resource is the sum of all atomic services that formulate the composite service.
- **Non-consumable Quantitative Attributes:** These attributes are used to limit the quantitative attributes for each single atomic service. The attributes can present either amount or scale.
- **Qualitative Constraints:** Those attributes which can't be expressed by quantities are called qualitative attributes. Qualitative Constraints are those qualitative attributes which specify the requirements to execute a web service.
- **Qualitative Facts:** Another kind of qualitative attributes, such as service type, service provider or geographical location, specify the facts regarding the services' environment. Those attributes can be regarded as goals in LL.

The different categories of non-functional attributes are presented differently in logical axioms. The non-functional attributes can be described as either constraints or results. The constraints and results of the services can be presented as follows:

- **The constraints to the service:**

$$\Delta_1 = Consumable^x \otimes NonConsumable^y \otimes ! Constraint$$

- **The results produced by the service:**

$$\Delta_2 = NonConsumable^y \otimes ! Fact$$

3.3 Example

Here, we illustrate the LL presentation of the temperature report service example where both functionalities and non-functionalities have been taken into consideration. The complete DAML-S description of this example can be found at `http://bromstad.idi.ntnu.no/services/TempService.daml`. For the sake of readability, we omit the namespace in the name of the parameters.

The available atomic web services in the example are specified as follows:

$$\Gamma = \begin{array}{l} NOK^{10} \vdash LengthCM \multimap_{cm2inch} LengthInch \\ CA\_MICROSOFT \vdash (LengthInch \otimes Brand \otimes Model \multimap_{ski} PriceUSD) \otimes LOC\_NORWAY \\ NOK^5, QUALITY^2 \vdash (PriceUSD \multimap_{usd2nok} PriceNOK) \otimes QUALITY^2 \end{array}$$

The formula presents three atomic services. *name2code* outputs the zip code of a given city. *temp* reports the Celsius temperature of a city, given the zip code of the city. *trans* transforms the Celsius temperature to the Fahrenheit temperature. NOK^{10} in the left hand side of the *name2code* service denotes that 10 Norwegian Krones(NOK) are consumed by executing the service. The service *trans* costs 5 NOK and has a quality level 2. The quality level is not a consumable value, so it appears at both the left and right hand sides. In the specification it is also said that the temperature reporting service *temp* is located in Norway and it only responses to the execution request that has certificated by Microsoft. For other attributes which are not specified in service specification, the values are not considered.

The required composite service takes a city name as input and outputs the Fahrenheit temperature in that city. It is specified by LL as follows:

$$\Gamma; \Delta_1 \vdash (LengthCM \otimes Brand \otimes Model \multimap PriceNOK) \otimes \Delta_2$$

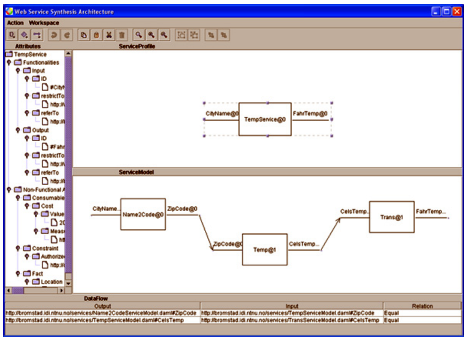
The non-functional attributes for the composite service are:

$$\begin{array}{l} \Delta_1 = NOK^{20} \otimes QUALITY^3 \otimes ! CA\_MICROSOFT \\ \Delta_2 = QUALITY^3 \otimes ! LOC\_NORWAY \end{array}$$

This means that we would like to spend no more than 20 NOK for the composite service. The quality level of all the selected services should be no higher than 3. The composite service consumer has certification from Microsoft ($!CA\_MICROSOFT$) and it requires that all location-aware services are located within Norway ($!LOC\_NORWAY$). ! symbol describes that we allow unbound number of atomic services in the composite service.

For the qualitative constraints (location), the service uses LOC_NORWAY to determine its value and we can determine in the set of requirements Δ_1 whether a service meets the requirement.

By now, we have discussed how DAML-S specification have been translated to LL extralogical axioms. Next step is to derive the process model from the specification of the required composite service. If the specification can be proven to be correct, the process model is extracted from the proof. we have stressed the proof in a separate publication [7] and therefore we don't go into detail here. The result dataflow of the selected atomic service are presented through a graphic user interface. A screen shot is presented in figure 2. In figure 2, the interface of the user required service is presented in the **ServiceProfile** panel (upperright) and the dataflow of the component atomic services is presented in the **ServiceModel** panel (lowerright).



Output	Input	Relation
http://bromstad.idi.ntnu.no/services/Name2CodeServiceModel.daml#ZipCode	http://bromstad.idi.ntnu.no/services/TempServiceModel.daml#ZipCode	Equal
http://bromstad.idi.ntnu.no/services/TempServiceModel.daml#CelsTemp	http://bromstad.idi.ntnu.no/services/TransServiceModel.daml#CelsTemp	Equal

Fig. 2. The Screen Shot

4 Composition Using Semantic Description

So far, we considered only exact match of the parameters in composition. But in reality, 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. 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.

In the following we extend the subsumption rule for both resource and goal. Here we should mention that the rules are not extension to LL. The subtyping can be explained by inference figures of LL. We write in following $\multimap_<$ to emphasis that these inference rules are for typing purposes, not for sequencing methods, when constructing programs.

First of all, the subtype relation is transitive.

$$\frac{\Sigma \vdash T \multimap_{<} S \quad \Gamma \vdash S \multimap_{<} U}{\Sigma, \Gamma \vdash T \multimap_{<} U} \; \textit{subtyping transitivity}$$

In addition, subsumption rules state the substitution between types.

$$\underset{\textit{goal subsumption}}{\frac{\Sigma \vdash T \quad \Gamma \vdash T \multimap_{<} S}{\Sigma, \Gamma \vdash S}} \qquad \underset{\textit{resource subsumption}}{\frac{\Sigma, S \vdash G \quad \Gamma \vdash T \multimap_{<} S}{\Sigma, \Gamma, T \vdash G}}$$

Such subtyping rules can be applied to either functionality (parameters) or non-functional attributes. Here we use two examples to illustrate the basic idea. First, let us assume that the output of the temperature reporting service is air temperature measured by Celsius scale, while the input of temperature translation service is all Celsius temperature. Because the later is more general than the former, it is safe to transfer the more specific output to the more general input. Another example considers the qualitative facts. If an atomic service is located in Norway, we regard Norway is a goal in LL. Because Norway is a country in Europe, it is safe to replace Norway with Europe. Intuitively, if the user requires a service that is located within Europe, the service located within Norway meets such requirement.

In this paper, we assume that the ontology used by service requester and that for the service provider are interoperable. Otherwise, the ontology integration is another issue which is beyond the scope of this paper.

5 Conclusion

This paper approaches the important issue of automatic semantic web service composition. It argues that Linear Logic, combined with semantic reasoning for relaxation of service matching (choosing), offers a potentially more efficient and flexible approach to the successful composition of web services. To that end, an architecture for automatic semantic web service composition is introduced. The functional settings of the systems are discussed and techniques for DAML-S presentation, Linear Logic presentation, and semantic relaxation are presented. A prototype implementation of the approach is proposed to fulfill the task of representing, composing and handling of the services. This paper concentrate on the automatic translation part and the semantic relaxation part, while the theorem proofing part has been stressed elsewhere[7].

Some works have been performed on planning based on semantic description of web services. In [5], the authors adapt and extend the Golog language for automatic construction of web services. The authors addressed the web service composition problem through the provision of high-level generic procedures and customizing constraints. SWORD[6] is a developer toolkit for building composite web services. SWORD uses ER model to specified the inputs and outputs of the web services. As a result, the reasoning is made based on the entity and attribute information provided by ER model. [8] presents a semi-automatic method for web service composition. The choice of the possible services are based on functionalities and filtered on non-functional attributes.

The main difference between our methods and the above methods is we consider the non-functional attributes during the planning. Usage of Linear Logic as planning language allows us formally define the non-functional characteristics of web services, in particular, quantitative attributes. In addition, we distinguish the constraints and facts in qualitative attributes. The planner treats them differently in logic formulas.

Also, as more and more organizations and companies embrace the idea of using web service interface as a cornerstone for future Grid computing architecture, the author hope that the revealing and discussing of semantic related issues will inform researchers in Grid computing of the intricate problem of service composition which might as well rise up in Grid service research.

Our current work is directed to add the disjunction connective to the logical specification of service output. This is useful when we should consider exceptions or optional outputs of atomic services. By using disjunction, the planner is also able to generate control constructs such as choice and loop. Although the introduction of disjunction is easy in logic presentation, the proving speed is slowed down significantly. The mechanism to improve the computation efficiency of proving is also under consideration.

References

1. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid. Online: http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf, January 2002.
2. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
3. Patrick Lincoln. Deciding provability of linear logic formulas. In *London Mathematical Society Lecture Note Series*, volume 222. Cambridge University Press, 1995.
4. David Martin et al. DAML-S(and OWL-S) 0.9 draft release. Online: http://www.daml.org/services/daml-s/0.9/, May 2003.
5. Sheila McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning(KR2002)*, Toulouse, France, April 2002.
6. Shankar R. Ponnekanti and Armando Fox. SWORD: A developer toolkit for web service composition. In *The Eleventh World Wide Web Conference*, Honolulu, HI, USA, 2002.
7. Jinghai Rao, Peep Kungas, and Mihhail Matskin. Application of linear logic to web service composition. In *The First International Conference on Web Services*, Las Vegas, USA, June 2003. CSREA Press.
8. 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*, 2002.