Formal Functional Description of Semantic Web Services: The Logic Description Method

Li Ye
Beijing University of Posts and Telecommunications
P.O. Box 187
Beijing, China 100876
sagi.ye@gmail.com

Junliang Chen
Beijing University of Posts and Telecommunications
P.O. Box 187
Beijing, China 100876
chjl@bupt.edu.cn

ABSTRACT
The semantic encoding of Web Service description is the foundation of automation. This paper proposes the Logic Description Method (LDM) which is formal, concise, and expressive. Service descriptions and knowledges (rules) are represented as first-order logic formulas which grasp the straightforward and natural semantics of them. ‘I’put, ‘O’utput, ‘R’elationship, ‘P’recondition, and ‘E’ffects are selected as the five essentials to define a service’s function. The ‘IORPE’ are all represented as predicates. Knowledge-base is used to gather all the definitions of knowledges and predicates. It build up a firm semantic environment where services have unambiguous functional definition. Furthermore, the semantic of a composed service can be easily inferred in first-order logic. So that it can then be verified with the semantic of the request to check for correctness. An example is set up to show how all the features of LDM works.

Categories and Subject Descriptors
F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs—Specification techniques; D.1.2 [Programming Techniques]: Automatic Programming; D.2.4 [Software Engineering]: Software/Program Verification—Correctness proofs, Formal methods

General Terms
Theory

Keywords
Semantic Web Services, Functional Description, First-Order Logic, Service Composition, Service Verification

1. INTRODUCTION
1.1 The Semantic Web vision
The Semantic Web [3], propagated by W3C, differs from today’s web in that it offers machine-processable web content. Thus make it possible for the use of variety intelligent techniques. For example, an intelligent financial agent can keep an eye on your stock prices if it can ‘understand’ the information on the web.

The key technologies of Semantic Web include, a) using XML instead of HTML to format the web contents so that the semantic of the information is carried with the meta-data essential; b) Ontologies where terms of the XML and relationships between these terms are defined; c) Logics, which allow new knowledges to be inferred from old knowledges through rules, to facilitate the intelligent features of agent; and d) Agents which profit from all aboves to take away the annoying works from human.

The essence of building a Semantic Web is to set up ontologies and define terms in them. Then using these terms to define the informations on the web. And encoding all these ontologies and webs in a way, e.g. XML, that can easily be processed by the computer.

1.2 The Web Services and Semantic Web Services
Besides the semantic tide, the web is also undergoing another evolution: from an information repository to a service provider. Web Services, a way to SOA (Service Oriented Architecture), makes the client and server talk in a more standardized way. Thus expand the component inter-operation from intra-domain to WWW-domain.

Semantic Web Services [11] are the result of the combination of these two evolutions. By Semantic Web Services, we mean services with an unambiguous and machine-processable description of their properties, interfaces, capabilities, and effects. This is achieved by using the ontology terms to describe the services.

2. MOTIVATION
2.1 Automation
Both the two trends of evolution share a common motive: to make the things more automatic. People will not be satisfied if the computers can only perform simple calculations or reactions. We wish them also to understand, to create, to make decision or even to think. Specially in the Web Service
world, the mechanisms to automate the task of discovering, invoking, composing, and verifying services are needed.

One of the main purposes of Semantic Web Services is to meet these requirements. The semantic encoding of service description is the foundation of automation. For example, an agent is able, a) with the interface description, to automatically communicate with the service; b) with the capabilities description, to understand what the service can do; and c) with the effects description, to know what will happen after the invocation.

The interface should tell what the service takes as input and what data it will produce as output. But this is not enough for the agent to understand the service. For example, it is almost nonsense only to tell an agent that a service takes an integer as input and produces another integer as output. The agent is lack of information to figure out what the service will really perform. A service increases the input by 1 and a service doubles the input will look no difference by their input/output data-types.

The capabilities description which defines the relation between the input and output, say IO-Relation, must be set to show the function of the service. In this way, the service doing increasing will be described to have an IO-Relation of "Successor(input, output)", which means the output is the successor( defined in the integer system) of the input, and the service doing doubling can be described to have the IO-Relation of "Double(input, output)", with the straightforward meaning.

Besides them, preconditions and effects are also important things that must be described. Preconditions stand for those situations which must exist for the service to be valid. These situations represent the states of the world (environment). If the preconditions do not hold, then invoking the service will fail or lead to an uncertain result. Effects are those things which will happen after the execution and change the environment. By environment we mean the world’s state which can not be represented by inputs and output. For example, the “deliver of goods” or “charge of bank account”.


2.2 An example: the air temperature service

Before laying down the method of formal functional description of Web Services, let’s look at a simple example about air temperature. All the services available to be used are listed in Table 1.

Now a mobile station user want to get the air temperature (in Fahrenheit unit) of his current location. An automatic agent should have the capability to solve this problem by composing the services defined above if he knows that:

1. An object carrier has the same location of this object;
2. A mobile station user carries a mobile station;
3. Weather information includes air temperature;
4. One temperature can be represented in different units while still has the same physical meaning.

These things are called knowledges. With (1) and (2), the agent can figure out that the location of the mobile station is same as the location of the user, if he/she carries the mobile station; with (3), he knows that once retrieved the weather information, the air temperature information is also got; and with (4), he has the confident of translating temperature between different units (by user prefer) while still remaining its original meaning. All these service descriptions and knowledges are needed to perform a successful automatic service composition task.

3. SOLUTION

3.1 The Logic Description Method (LDM)

Our solution to the above problem is to encode both services and knowledges into logic. Services can be encoded in a first-order logic (FOL) formula template:

\[ \forall x_1 \ldots x_n (T1(x_1) \land \ldots \land Tn(x_n) \land P(x_1,\ldots,x_n) \rightarrow (\exists r (Tr(r) \land R(x_1,\ldots,x_n,r)))) \]

where

1. \( x_1 \ldots x_n \) : represents the inputs of the service;
2. \( r \) : represents the output of the service;
3. \( T \ast (k) \) : means parameter ‘k’ has the data-type of ‘T’;
4. \( P(x_1,\ldots,x_n) \) : means “\( x_1,\ldots,x_n \)” satisfy the precondition ‘\( P \)’ which is defined as an knowledge in the knowledge-base;
5. \( Result(x_1,\ldots,x_n, r) \) : Result of the service execution. A composition of the IO-Relation and the effects. Defined as an knowledge in the knowledge-base also.

It could be read as:

given the input list of certain types, once the precondition holds, we can get an output which satisfies a certain relationship with the inputs and be sure of the rise of certain effects.

This first order formula contains all the needed informations of a Semantic Web Service description we defined above. The input and output is the abstract interface of the service, which can further be mapped to a more concrete binding specification such as WSDL [7]. The type of each parameter (both input and output) is specified by a 1-arity predicate with its name as the data-type’s name. The “\( \forall \text{input} (\exists \text{output}) \)” schema means that for every valid input, an output value can be drawn by calling this service (or say applying this formula).

The ‘Result’ tells all the information about the function of the service. It includes the information transformations the service will commit and the effects which will happen by calling this service. An information transformation is the IO-Relation. For example, a simple service which doubles an input integer value will has the ‘Result’ defined as ‘Double(input, output)’. Effects are other things happened when executing the service that can not be represented by input and output. For example, a book selling service might requires an input of a bank account the user owns and the book that he wants to buy. And the output might be just a confirmation number. The things really happened behind this can be represented as effects, which in this scenario might be the charge of the account and the delivery of the
3.2 Using LDM to encoding the air temperature example

Let’s take the air temperature example to see how LDM works. Several predicates (type definitions and relation verbs) need to be defined first, they make up the basic of the knowledge-base. See Table 2 for details.

For simplicity, all the services in this example have no preconditions and effects. They could be encodes as,

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TempC2F</td>
<td>convert a temperature in Centigrade to Fahrenheit</td>
</tr>
<tr>
<td>TempF2C</td>
<td>convert a temperature in Fahrenheit to Centigrade</td>
</tr>
<tr>
<td>Weather</td>
<td>supply weather information of a certain area</td>
</tr>
<tr>
<td>Location</td>
<td>supply location information based on a mobile station</td>
</tr>
</tbody>
</table>

And the knowledges are defined as:

∀ w, tc l (WeatherTemp(w, tc) & WeatherInfo(l, w)) → TempInfoC(l, tc).

The air temperature extracted from the weather data tells information of the same location where the weather data is about.

∀ l, tf (TempInfoF(l, tf) & C2F(tf, tf)) → TempInfoF(l, tf).

∀ l, tf (TempInfoF(l, tf) & F2C(tf, tf)) → TempInfoF(l, tf).

The correspondence between air temperature represented in Centigrade and Fahrenheit.

Look at knowledge (6) and (7), these two have the similar form of a service encoding. That is the “∀ input (\(\exists output\))” schema. It is because they share the same functionality as services which can perform an information transformation. That is from one kind of information, there exists another kind of information which has a certain relation with the original one. They have the “input → output” schema. So it is straightforward to treat these knowledges as services (tag it with a service name) by an agent while performing a composition task. By this philosophy, the knowledge (6) is tagged as ‘GetMS’ and the knowledge (7) is tagged as ‘GetTemp’.

Since what a user request is also a service, it can be encoded almost the same way. The kind of service the user request in the air temperature example could be encoded as:

∀ user (MsUser(user)) → (\(\exists tf (Fahrenheit(tf) \& R(user, tf))\)).

And the knowledge about R(user, tf) is defined in the knowledge-base as:

∀ user tf (\(\exists l (At(user, l) \& TempInfoF(l, tf))\) → R(user, tf)).

This means the service accepts an identifier of a mobile station user and returns a temperature in Fahrenheit unit, where the temperature is the air temperature of the location where the user is at now. With these definitions, the agent should be able to infer that the requested service can be composed as,

\(\forall w, tc l (WeatherTemp(w, tc) \& WeatherInfo(l, w)) → TempInfoC(l, tc)\). (8)

\(\forall l, tf (TempInfoF(l, tf) \& C2F(tf, tf)) → TempInfoF(l, tf)\). (9)

\(\forall l, tf (TempInfoF(l, tf) \& F2C(tf, tf)) → TempInfoF(l, tf)\). (10)

Table 1: All services used in the “air temperature” example.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
### 3.3 Advantages and disadvantages

LDM has its advantages. First, it is based on the first-order logic, which is already a reliable, mature, and well studied mathematics system. It is formal, precise, and concise. The services and knowledges are encoded with their original semantics. This makes them self-explaining. By reading the meaning of the formula in first-order logic, the semantic of the service/knowledge is automatically explained. There is no need to define another set of notations combined with their own semantics. Everything is just logic, and only need to be explained in the logic way.

Second, the composed services have firmly and precisely inferred semantics. This semantic can be deduced from the semantics of the services and knowledges which comprise it. It is determinate and explainable. If an agent has worked out a result, it can be easily checked for correctness by comparing its semantic with the requested semantic. This is the key feature for the usage of automatic mechanism in a composition task because it makes the result believable. Otherwise, a complex algorithm must be designed to prove the correctness of the result when the algorithm might became too complex to be proved. But with the nature of logic, once the hypothesis stands and the deduce methods are correct, the result is also correct. So the burden of correctness check is greatly released.

Take the result of the “air temperature service” for example, it can be explained as:

1. At the beginning, the input is user which has type of MsUser(user);
2. Let $ms = GetMS(user)$. —[by knowledge (6)]
   Then $ms$ is a mobile station which the user carries which means Carries(user, ms) (a).
3. Let $l = Location(ms)$. —[by service (4)]
   Then $l$ is the location of $ms$, which means At(ms, l) (b).
4. Let $w = Weather(l)$. —[by service (3)]
   Then $w$ is the weather information of area $l$, which makes WeatherInfo(w, l) (c) holds.
5. Let $tc = GetTemp(w)$. —[by knowledge (7)]
   This extracts the air temperature ‘tc’ from ‘w’ and makes WeatherTemp(w, tc) (d) holds.
6. Let $tf = Temp2CF(tc)$. —[by service (1)]
   This converts the temperature $tc$ into Fahrenheit unit and keeps the meaning of $C2F(tc, tf)$ (e).
7. From (c) and (d), infers TempInfoC(l, tc) (f). —[by knowledge (8)]
8. From (f) and (e), infers TempInfoF(l, tc) (g). —[by knowledge (9)]
9. From (a) and (b), infers At(user, l) (h). —[by knowledge (5)]
10. From (g) and (h), infers R(user, tf). —[by knowledge (12)]
   This matches the specification of the requested service. This means the composed service matches the requested service.

Third, in virtue of the powerful representing ability of FOL, LDM is rather expressive. The predicates and the knowledges together composed the knowledge-base. The Semantic Web rely on ontologies to define its semantics. The ontology itself is commonly based on the theory of description logic(DL) which has the much less express powerful

| Table 2: All predicates used in the “air temperature” example. |
|------------------|------------------|
| **Predicate**    | **Semantic**     |
| Centigrade(x)    | $x$ is a value of temperature in Centigrade unit. |
| Fahrenheit(x)    | $x$ is a value of temperature in Fahrenheit unit. |
| Loc(x)           | $x$ is a value of location data. |
| WeatherData(x)   | $x$ is a value of weather data. |
| MobileStation(x) | $x$ is the identifier of a mobile station. |
| MsUser(x)        | $x$ is the identifier of a mobile station user. |
| C2F(x, y)        | $x$ in Centigrade unit and $y$ in Fahrenheit unit represent the same physical temperature. |
| F2C(x, y)        | $x$ in Fahrenheit unit and $y$ in Centigrade unit represent the same physical temperature. |
| WeatherInfo(x, y)| Location $x$ has the weather information of $y$. |
| At(x, y)         | Object $x$ is at the location of $y$. |
| Carries(x, y)    | $x$ carries the object $y$. |
| TempInfoC(x, y)  | $y$ is the air temperature in Centigrade unit extracted from the weather data $x$. |
| TempInfoF(x, y)  | $y$ is the air temperature in Fahrenheit unit of location $x$. |

**Figure 1:** Composed Service of “air temperature” example.
than FOL. The knowledge-base of LDM is like the ontology of Semantic Web which based on FOL instead of DL. All the terms (predicates) used to build an knowledge or a service description should already have been defined in the knowledge-base.

Finally, the two parts of the service’s function, information transformation and effects, are encoded in a unified method which is the ‘Result’ part in the service encoding template. Together with I/O type definitions and preconditions, they are totally merged into one single formula. The air temperature example did not illustrate the use of preconditions and effects. But since they have no particularity against the I/O types and relationship specifications (they are all just predicates in logic), the methods used to deal with them should be no different. This will significantly simplify the automatic agent’s processing effort.

Everything has its two sides. As a prototype method, LDM has only been applied on several simple examples to show its usability. The real world is far more complex. LDM only deals with the ‘IORPE’ of the Web Services which are essential but not all. Those aspects such as the performance, cost, reliability, and availability are not included.

In spite of all the merits it brings, FOL is a complex and heavy logic system compared to DL. This may adds computational cost and limit the size of problems an agent can handle. The knowledge-base might be another barrier for the application of LDM. LDM need an knowledge-base which is congruent and uniform. Every term can only be given one semantic. While in the real world, it is always hard to find such an knowledge-base. In fact, none of the knowledge representing systems including the ontology used by Semantic Web can by pass this barrier. It is still a hot researching topic.

4. RELATED WORKS

4.1 In the Web Service domain

OWL-S [10], the successor of DAML-S, is one of the most popular Semantic Web Service description method. It is an upper ontology for services. All ontologies, each corresponds to a service, inherited from OWL-S have the same common structure and information categories. An OWL-S ontology is composed of 3 parts, which are 1) Service Profile what it does, 2) Service Model how it works, and 3) Service Grounding how to access it.

Here, we concern the functional description part only. That is the main content of the Service Profile of OWL-S. It specifies the (i)inputs required by the service and the (o)outputs generated, and describes the (p)reconditions required by the service and the expected (e)ffects that result from the execution of the service, which abbreviated to ‘IOPE’. They represent the two aspects of the functionality of the service: the information transformation (represented by inputs and outputs) and the state change produced by the execution of the service (represented by preconditions and effects).

The IOPEs are formally defined in order to enable machine-processing. Other informations in the Service Profile such as the ‘Service Name’, ‘Contacts’, and ‘Text Description’ are defined in a rather informal way which are mainly for human reading. So the IOPE is all what an automatic agent can depend on to figure out the function of a service. (There are techniques [4] to ‘guess’ the function of a service from the name of the service. But that depend too much on the intelligent of the agent and is rather difficult to get a faithful result.)

As a w3c standard, OWL-S’s functional description is much more detailed and canonical. But the essence is almost the same. The difference is between ‘IOPE’ and ‘IORPE’. OWL-S’s ‘IO’ section only defined the data-type of them. It has no concern of the IO-Relation, that is the ‘R’ part in LDM. This is just what the LDM emphasis on. OWL-S do has the ability to make up this in its “result” section. Still it is not a required information and is far away from OWL-S’s center attention. Further more, due to the lack of ‘R’ part, it is hard to verify the correctness of a composed OWL-S service. The different methods used to define ‘IO’ and ‘PE’ are also more difficult to be processed compared to the unified method used in LDM.

In [12], Web Services are encoded in Linear Logic(LL) [6] and then passed to a LL prover to perform the automatic composition task. In spite of the aditional features LL provided, the method used in [12] only touched the input/output parameters data-type. No preconditions, effects, and IO-Relation are concerned.

In [8], a method was proposed to define a Web Service with “behavioral models”. That is to describe services by executable specification behavioral models. The discovery and validation methods are also based on the behavioral models. This holds a much different understanding of a service description from OWL-S or LDM. And there might be something in this method which LDM can use for reference.

4.2 Software specification

The idea of LDM comes from the method used in the programing synthesis domain to represent the specifications of programs or software components. [9] defined the specification as the relationship between the input and output. FOL is used as the language of specification. And the predicates used in the FOL formulars are to be explained based on some background theories.

The similarity of the methods comes from the similarity the problem domain. Web Service is just one kind of software component in its nature. It is special because of supposed to be well defined so that can be searched and invoked through an open interface. The LDM uses the common template of service encoding and the common predicates defined in the knowledge-base to match this characteristic. A common template of service encoding makes the definition much more easier to be translated into other format such as XML. Thus facilitate the transfer and process of the service description between agents. The knowledge-base is a combination of the ontology used in the Semantic Web domain and the background theory used in [9].

5. FUTURE WORKS

The LDM is only a prototype method now. Before it can be applied to real use, there is still a lot of things to do.

In order to take advantage of the quite powerful language of XML, an XML encoding method need to be defined to convert the description into XML. Everything the LDM produces is FOL. And there are already many methods proposed to encode FOL in XML. So, this work should be done without too much effort.

For the compatible concern with OWL-S, the most popular Semantic Web Service definition standard, a tool is
planned to do the conversion between them. As only a functional description method, LDM lacks all other informations such as grounding and nonfunctional properties required in OWL-S. Since OWL-S already has a well defined schema for these parts, we preferred to use them directly and to only process the functional part when converting between them. From LDM to OWL-S, it might be a difficult in converting the ‘R’ and ‘E’ parts. These two parts are combined in LDM without difference on the surface. Our first reflection is to rely on the knowledge-base to tell us what predicates stands for effects which describe the changes of the world’s state. Also, there will be problems from OWL-S to LDM. The major one is that OWL-S misses the ‘R’ part which is highly required by LDM. However, it might still works. Only that the services described without the ‘R’ part are not likely to have any chance of being used by an automatic agent because no agent could figure out their function.

Our ultimately object of this research is to try solving the problem of automatic service composition and verification. LDM is the basic of these researches. It has turned all the problems in Web Services domain into the well studied FOL domain. There are various logic related tools, such as proof assistants [5, 1], automatic theorem prover (ATP) systems [2], expert systems, etc. All these tools may be used to facilitate the construction of our automatic system.

6. REFERENCES