Approximating Mathematical Semantic Web Services Using Semantic Description Decomposition and Approximation Formulas

Andrei-Horia MOGOS, Adina Magda FLOREA
andrei.mogos@cs.pub.ro , adina.florea@cs.pub.ro
"Politehnica" University of Bucharest, România

The domain of semantic web services is one of the important areas of the computer science research. Mathematical semantic web services are very useful in practice, however only a small number of related research results are reported. This paper presents a system that attempts to obtain a mathematical semantic web service using its semantic description. If the service cannot be found, then the system tries to obtain an approximation of that service, using a semantic description decomposition and approximation formulas.

Keywords: *mathematical semantic web service, web service approximation.*

1 Introduction

A web service is "a software system designed to support interoperable machine-to-machine interaction over a network." [1] In order to support important dynamic and automated tasks such as discovery, selection, and composition, web services must be supplied with more semantics. The semantic web, that is a web of machine-processable information, could solve this problem by combining web service technology with the semantic representation of information and, through this, enable automatic and dynamic interaction between software systems [2].
Recently, there are some efforts to develop languages for the description of mathematical semantic web services. The most important initiative is the MONET project [3]. The aim of the MONET project is to demonstrate the applicability of the latest ideas for creating a semantic web to the world of mathematical software, using algorithms to match the characteristics of a problem to the advertised capabilities of available services and then invoking the chosen services through a standard mechanism.
The paper is organized as follows. In Section 2, we give a description of the problem. In Section 3, we describe the algorithm used to solve the problem. Section 4 presents the structure of the system. In Section 5, we give an example of how our system works while Section 6 contains some conclusions and future work.
In the rest of the paper, we will use the abbreviation MSWS for 'mathematical semantic web service'.

2. The problem

A MSWS *m2* approximates another MSWS *m1*, if for each input of *m1*, *m2* gives almost the same output as *m1*. The difference between the outputs of *m1* and the outputs of *m2* is controlled by an expression error.
We understand by semantic description (SD), the semantic information used for a web service, i.e., that information that allows a machine to understand what a service does. We understand by complete description (CD), the entire information used for describing a web service, including its semantic description.
The problem we are trying to solve can be described as follows: we have a semantic description of a MSWS, let's call it service ***D***, a library of MSWSs and another library of approximation formulas. We want to provide the service ***D***, or at least an approximation of that service.
The initial semantic description of the MSWS, the semantic descriptions of the MSWSs in the library, and the approximation formulas use the same semantic language. In this way, we can see the right part of an approximation formula, ignoring the expression error, as a semantic description of a MSWS ***App***. Therefore, we can construct that MSWS ***App*** using the same algorithm as we use for the initial MSWS, service ***D***.
We make the following convention: every MSWS needed for constructing a more com-

plex MSWS necessary for an approximation formula can be found in the MSWS library.

3. The algorithm

Next, we will describe the algorithm used to solve the problem:

```
 1: Algorithm (SD_MSWS)
 2: CD_MSWS = search (SD_MSWS, MSWS_library)
 3: if CD_MSWS < > void
 4:   return CD_MSWS
 5: d_tree = void
 6: d_tree = decompose (SD_MSWS, d_tree)
 7: tree_level = 1
 8: while (1) {
 9:  aux_tree_level = 0
10:  for-each node in d_tree, level =
        tree_level {
11:    SD_MSWS_node = get_SD_MSWS (node)
12:    CD_MSWS_node = search (SD_MSWS_node,
                       MSWS_library)
13:    if CD_MSWS_node < > void
14:      add (CD_MSWS_node, node)
15:    else {
16:      d_tree = decompose (SD_MSWS_node,
                              d_tree)
17       aux_tree_level = 1 }}
18:      if aux_tree_level = 0
19:        break
20:      else
21:        tree_level = tree_level +1 }
22:      for-each leaf in d_tree {
23:        CD_MSWS_leaf = get_SD_MSWS (leaf)
24:        if CD_MSWS_leaf = void {
25:          SD_MSWS_leaf =get_SD_MSWS (leaf)
26:          SD_approx_formula = search
       (SD_MSWS_leaf, approx_formula_library)
27:          if SD_approx_formula = void
28:            return ApproxFailed
29:          CD_MSWS_leaf = Algorithm
                      (SD_approx_formula)
30:          add (CD_MSWS_leaf, leaf) }}
31: CD_MSWS = compose (d_tree)
32: return CD_MSWS }
```

The algorithm receives a semantic description of a web service and returns the complete description of that web service.

If we cannot find a MSWS in MSWS library, corresponding to that semantic description, we decompose that description until we obtain a tree: each leaf of this tree contains a semantic description that corresponds to a MSWS in MSWS library or contains a semantic description that doesn't correspond to a MSWS in MSWS library, but the semantic description cannot be decomposed any further. The main idea used for a decomposition step is the following: we have a mathematical calculus formula; thus, we have an operation order, so at each step a single operation is computed. As our point of decomposition, we consider the operation with the lowest priority. If the operation is *f (f1 (n), f2 (n), f3 (n), … fm (n))*, the decomposition is *f, f1 (n), f2 (n), f3 (n), …, fm (n)*. If an operation *fi (n)* cannot be decomposed, it will appear as '*fi*' in the decomposition, instead of *fi (n)*. Here '*n*' is the input of the MSWS.

For each leaf in the tree, that doesn't have a corresponding MSWS in MSWS library, we try to find an approximation formula. If we find a formula, then we search in the MSWS library to get some MSWS that, composed, computes the right part of the approximation formula. After this step, the tree contains only leaves with associated MSWSs.

The call of the function '*Algorithm*' in line 29, doesn't create an infinite recursion, because the function is called with the argument 'SD_approx_formula' and from the convention made in Section 2, all the necessary MSWSs for an approximation formula in Approximation Formula Library will be found in the MSWS Library.

The last step is to compose the complete descriptions of the MSWSs in the tree to get the complete description for the initial service. This complete description gives us, among other information, the information we need to call the service.

4. System Structure

The system structure is presented in Figure 1. The Processing Agent receives a semantic description of a MSWS and returns a complete description of that MSWS. The agent uses the algorithm presented in Section 3 and the interactions with Decomposition Module, Composition Module and Searching Agent. The Decomposition Module is responsible with the function '*decompose*' from the algorithm, in lines 6 and 16. The Composition Module is used for the function '*compose*' in line 31.

The Searching Agent is used for the function '*search*', in lines 2, 12, 26. For the '*search*' function call in lines 2 and 12, the Searching Agent uses the MSWS Matching Module. This module receives a semantic description of a MSWS and tries to find a corresponding MSWS in the MSWS Library. For the '*search*' function call in line 26, the Searching Agent uses the Approximation Formula

Matching Module. This module receives a semantic description of a MSWS and tries to find a corresponding approximation formula in the Approximation Formula Library.

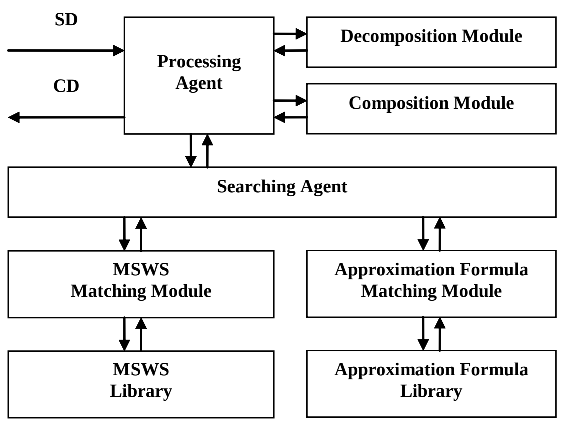

Fig.1. The system structure

5. An example

To use our system, we must use a semantic language which allows every mathematical formula to be written using only ASCII characters (with codes 0-127).

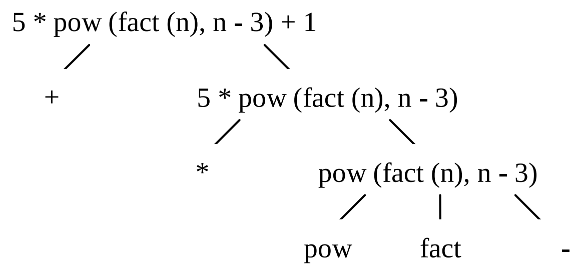

Fig.2. The operations tree

We have the semantic description *5 * pow (fact (n), n - 3) + 1*, where *fact (n)* stands for *n!*. We consider that this MSWS has one input '*n*' and one output. The MSWS Library contains the following MSWSs: +,-, *, /, *pow*, *sqrt*. The Approximation Formula Library contains, among other formulas, the following formula:

fact (n)≈ sqrt (2*n*π) * pow (n, n) * pow (e, -n) (Stirling's formula [4])

In Figure 2, we present the tree constructed for this example by our algorithm. All the leaves in this tree have a corresponding MSWS in MSWS Library, except for '*fact*'. For '*fact*' we find an approximation formula in the Approximation Formula Library. For the expression *sqrt (2*n*π) * pow (n, n) * pow (e, -n)*, in the Stirling's Formula, we apply again our algorithm. The resulting tree has for each leaf a corresponding MSWS in MSWS Library. Finally, our expression is

5 * pow (sqrt (2*n*π) * pow (n, n) * pow (e, -n), n - 3) + 1,

and after the composition process, we have enough information to access a MSWS that performs the computations specified by the above expression.

6. Conclusions

This paper presents an original method of approximating MSWSs using semantic description decomposition and approximation formulas. The approach used for this purpose is the composition of semantic web services using goal decomposition. Our system tries first to use some MSWSs in a MSWS Library, and if this fails, it uses some approximation formulas to create the composed MSWS.

One problem that has to be further considered is errors propagation. The approximation formulas might be very good, but depending on the structure of the expression corresponding to the initial MSWS, the overall error can become big enough so that the approximation of the MSWS cannot be useful in practice. Another problem to consider is the construction of a metric to compare two approximation formulas for the same function. This is not easy because it might be necessary to compare the errors for the two formulas, which can lead to the necessity to perform function comparison.

7. References

1. W3C Working Group: "Web Services Architecture", February 2004, http://www.w3.org/TR/ws-arch/
2. Studer, R., S. Grimm, and A. Abecker (Eds.): *Semantic Web Services. Concepts, Technologies, and Applications*, Springer, 2007, p. 159
3. The MONET Project, http://monet.nag.co.uk/monet/
4. planetmath.org/encyclopedia/StirlingsApproximation.html