Journal of Universal Computer Science, vol. 18, no. 11 (2012), 1576-1597
submitted: 30/11/11, accepted: 15/5/12, appeared: 1/6/12 © J.UCS

Towards a Practical Solution for Data Grounding in a Semantic Web Services Environment

Miguel García Rodríguez
(Department of Computer Science, University of Oviedo, Spain
miguel.garcia@weso.es)

Jose María Alvarez Rodríguez
(Department of Computer Science, University of Oviedo, Spain
josem.alvarez@weso.es)

Diego Berrueta Muñoz
(R&D Department, Fundación CTIC, Gijón, Spain
diego.berrueta@fundacionctic.org)

Luis Polo Paredes
(R&D Department, Fundación CTIC, Gijón, Spain
luis.polo@fundacionctic.org)

Jose Emilio Labra Gayo
(Department of Computer Science, University of Oviedo, Spain
jelabra@weso.es)

Patricia Ordoñez De Pablos
(Department of Business Management, University of Oviedo, Spain
patriop@uniovi.es)

Abstract: Grounding is the process in charge of linking requests and responses of web services with the semantic web services execution platform, and it is the key activity to automate their execution in a real business environment. In this paper, the authors introduce a practical solution for data grounding. On the one hand, we need a mapping language to relate data structures from services definition in WSDL documents to concepts, properties and instances of a business domain. On the other hand, two functions that perform the lowering and lifting processes using these mapping specifications are also presented.

Key Words: service-oriented architectures, web services, cloud computing, semantic web, ontologies, data grounding, semantic web services, interoperability, mapping languages
Category: H.3.5, G.2.2, I.2.4

1 Introduction

Web services are the base technology for Service Oriented Architectures (SOA) on the Web and the Cloud Computing [Taniar et al. 2011] realm. According to the architecture and definitions by W3C [Booth et al. 2004], a web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format, specifically using the Web Service Description Language (WSDL). Other systems interact with the web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

However, practical deployment of SOA architectures usually faces problems of integration due to the heterogeneity of the services, in particular, integration of different data models. Semantic Web Services (SWS) [Akkiraju et al. 2005, Battle and Bernstein 2005, de Bruijn et al. 2005, Martin et al. 2004] combine current web services and the semantic web technology [Anicic et al. 2006, Roman et al. 2006]. More specifically, SWS propose ontologies as common data models to abstract the definition of services. Consequently two different levels of service description appear [Burstein et al. 2005, Cabral et al. 2004]: some tasks can be automated at the semantic level, e.g. discovery [Álvarez Sabucedo and Rifón 2010, Pan et al. 2011] of services, while others can only be performed combining the semantic and syntactic descriptions of the services (e.g. invocation). The latter require a mechanism to translate data between these two levels.

"Data Grounding is the bidirectional process that downgrades a semantic model to a syntactic level through a subprocess called lowering and upgrades a syntactic model to a semantic level through a subprocess called lifting, enabling actual invocation of web services in SWS environments [see Fig. 1]."

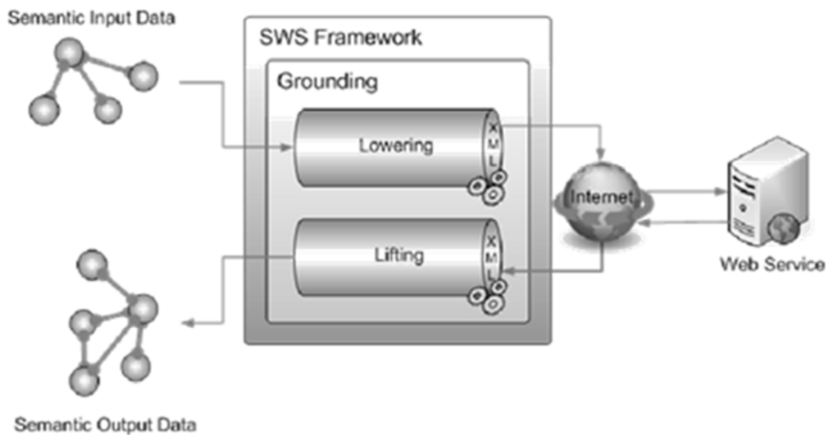


Figure 1: Grounding Process

We focus on data grounding [Kopecký et al. 2006] because it is the cornerstone to deploy semantic web services in production environments. Data grounding allows building a request from the information available in the semantic model and to process the response from the service. In this paper, we present a new approach to data grounding based on a declarative mapping language. We address the problem from a structural point of view, i.e., independently from the logical foundations of ontologies. Moreover, our proposal is restricted to the vertical transformations between different description levels of services. Horizontal transformations such as data mediation are assumed to be solved by other components of the SWS platforms; therefore they are out of the scope of this paper.

This article describes a new approach to perform data grounding subtasks comprised of a language mapping and a transformation process. The proposal is based on a direct mapping language M [see Fig. 2] between the syntactic description of the messages of a web service (XML Schema Fallside and Walmsley [2004] inside WSDL) and the graph-based semantic model built with ontology languages like RDFS(S), the W3C Ontology Web Language (OWL) or the Web Service Modeling Language (WSML). These mappings are clear instructions on how to transform between the two different description levels. Such transformations take place when a semantic agent needs to exchange information with a web service. In order to carry out this transformation, two main functions have been implemented with the objectives of: 1) generate XML content for SOAP requests and 2) interpret XML content of SOAP responses to create a graph. In the following sections both, the language mapping and the transformation processes are detailed. Furthermore, a real implementation and application of this proposal is presented.

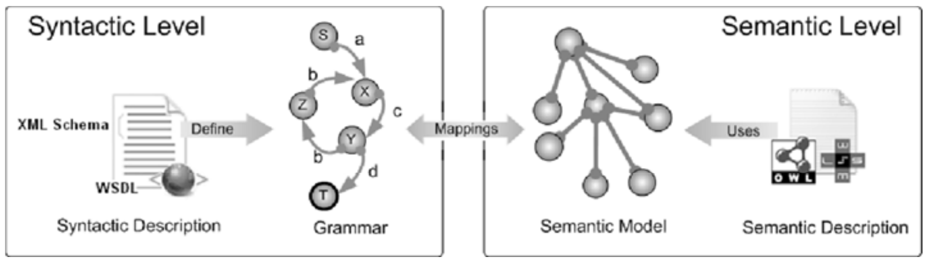


Figure 2: Direct mapping language between descriptions

2 Related work

Taking into account the current web technological stack, the problem of data grounding in semantic web services [Pedrinaci and Domingue 2010] consists on a structural transformation between XML trees (SOAP Messages) and RDF graphs (ontological data). Trees are special kinds of graphs, therefore algebraic approaches to graph transformation, such as the double-pushout (DPO) and the single-pushout (SPO), can be used [Corradini et al. 1996]. Although these techniques are a feasible solution from the formal point of view they are too complex from a practical point of view. The particular structure of XML trees and RDF graphs make it possible to devise specific solutions.

According to [Roman et al. 2006] there are three approaches to data grounding [see Fig. 3]:

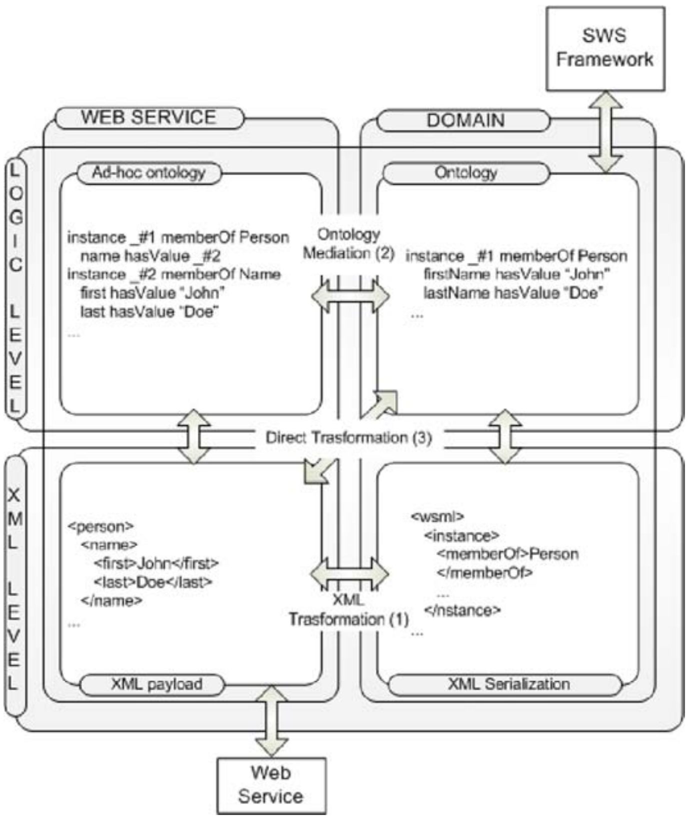


Figure 3: Approaches to Data Grounding. The upper half represents data in WSML ontologies

1. Transformations at the XML level are the first option to travel between a semantic model (serialized as XML) and a syntactic representation. The main advantage is that this process is based on existing and robust technologies: XSLT or XQuery, besides the serialization of a semantic model to XML is simple, a priori (e.g. RDF/XML serialization of a RDF graph). Nevertheless, there is a lack of homogeneity. Existing ontology languages like RDF, OWL or WSML-DL, etc. are based on graphs and they have different ways of being serialized as XML. Mainstream research is focused on this approach to implement grounding [Kopecký 2005, Pantschenko et al. 2005], e.g. WSMO initiative [Kopecký et al. 2007].
2. Transforming at the ontology level involves building an ad-hoc pseudo-ontology from the XML Schema of WSDL [Bohring and Auer 2005]. The grounding is implemented using ontology mediation [Mocan and Cimpian 2005] between that pseudo-ontology and the domain ontology. This method is based on merging, aligning and mapping techniques [Bruijn et al. 2006]. The map-pings are used as a set of rules that are executed on a rule engine, e.g. FLORA-2. The key point of this approach lies on the reliability and expressiveness of the built pseudo-ontology. Anyway, tools available to realize

this approach are still in an early stage of maturity.

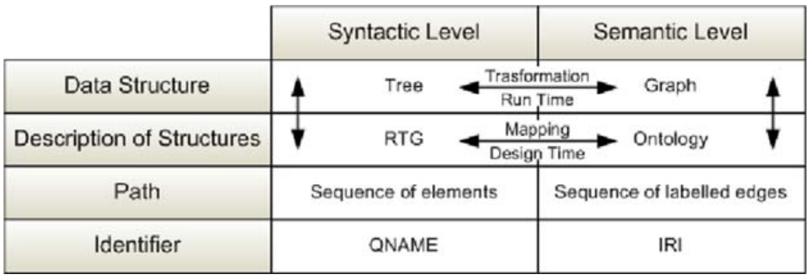
3. Direct transformations between elements from the XML document to entities belonging to a domain. There are two possible approaches: (a) ad-hoc transformation code that must be manually programmed in a language such as Java or XSPARQL [Kopecký and Schütz 2008]; (b) using declarative map-pings rules between XML Schema and the semantic model. In the second case, the mapping rules are written in a particular language and it is necessary to implement a processor to perform the transformations. In this field, the Semantic Annotations for WSDL and XML Schema [Farrell and Lausen 2007, Klímek and Necaský 2011] (SAWSDL) initiative and its predecessor WSDL-S [Akkiraju et al. 2005] are examples of this approach, but the implementations are not yet available or are committed to a particular project. Alternatively, OWL-S [Balzer and Liebig] extends both WSDL and OWL to specify grounding.

All described approaches require human intervention at design and validation time. In the first two options the human intervention is more complex than in the third one. In the first case, transformations are coded into programs [Fernández et al. 2010], in the second one mapping rules are declared via rules in complex representational languages. Therefore we finally chose the third option because it is more suitable for people not trained [García et al. 2011] in programming or knowledge engineering. It only needs human intervention to directly map entities with XML elements at design time but knowledge about technologies (e.g. XSLT or Flora2), operations (e.g. merging or aligning) or algorithms (e.g. PROMPT or GLUE) are not required.

3 Method to transform trees into graphs: use case for Grounding

As defined in previous works [Rodríguez et al. 2009], data grounding is the process of transforming data from the syntactic level to the semantic representation and back. In order to bridge the gap between syntactic and semantic levels, we need some kind of information that describes how the semantic data can be represented in XML and how the XML data returned from the service can be interpreted using its semantic description.

Our approach for data grounding is based on a direct mapping language that uses mapping rules to link data structures of WSDL documents with a sequence of IRIs from the ontologies. These mapping rules are clear instructions to perform transformations between XML trees [Cowan and Tobin 2004] and semantic data graphs. In our approach, we do not need the logic formalism of O to implement the grounding process, it is merely a structural process that executes a set of mapping rules between non-terminals symbols in a RTG grammar [Comon et al. 2007, Murata et al. 2001] (elements in XML) and the structure of objects descriptions [see Fig. 4]. At design time, mapping rules between the RTG grammar (extracted from XML Schema) and the domain ontology are created to link the syntactic and semantic descriptions. At run time, these mapping rules are applied in order to perform lowering and lifting operations. To support the transformation at run time, only the mapping rules and the RTG derived from the WSDL description is required. In other words, the semantic description of the web service and the T -Box of the domain ontology are not used during the actual transformation.



	Syntactic Level	Semantic Level
Data Structure	Tree ⟵ Trasformation / Run Time ⟶	Graph
Description of Structures	RTG ⟵ Mapping / Design Time ⟶	Ontology
Path	Sequence of elements	Sequence of labelled edges
Identifier	QNAME	IRI

Figure 4: Relation between syntactic and semantic representations

3.1 Mapping language

We propose a mapping language to realize data grounding. Our approach is declarative in the sense that we focus on what data must be transformed and not how to transform it. Let G = (NT,∑,S,R) be a regular tree grammar (RTG), extracted from the syntactic description of a web service (WSDL) [see Fig. 5]. Let O be a domain ontology used in the semantic description of the same web service. A tuple m = (ctx, e, α) is a mapping for grounding, where e ϵ NT, ctx ϵ NT (context of e) and ϵ IRI+. The mapping is the glue between the semantic and syntactic description. Non-terminal symbols of the grammar are mapped to paths in the graph. The context of a non-terminal symbol n is the derivation tree from the initial symbol S to n according to the production rules in the grammar. As [Rodríguez et al. 2009] indicates, there is a bijective correspondence between production rules and non-terminal symbols. Therefore, the sequence of production rules that de ne of a non-terminal symbols can be represented using a sequence of the non-terminal symbols from the initial symbol S. As we have reviewed [Rodríguez et al. 2009] our non-terminal symbols are named after the XML elements defined in the XML Schema and those are identified by QNames, therefore we can identify the non-terminal symbols using QNames. Consequently, the context ctx can be described by a sequence of QNames. The language of mapping rules for the grounding is defined by the following expression:

$$M\text{: QName* x QName x IRI}^{+}$$

3.2 Transformation operations

There are two main scenarios when data grounding runs: 1) Lowering, the map-ping rules guide the process through the semantic model to extract the parameters of the precondition that are used to create the SOAP body of the request. The mapping language supports several entities of the semantic model being mapped to the same non-terminal symbol. Lowering is restrictive with the generated output in order to ensure the creation of valid and well-formed XML documents. 2) Lifting, the mapping rules guide the process of parsing the SOAP content and building a set of instances according to postconditions (described using a semantic model) of the web service.

3.3 $L_{owering}$ function

Given a regular tree grammar, G, a semantic request Drq, and a set of mappings M, we de ne the lowering process as the transformation function of the semantic information, starting in the node v, to a syntactic representation T(∑).

$$L_{owering} : D_{rq} \times G \times M \times v \rightarrow T(\sum)$$

$NT = \{NT_{fare}, NT_{owner}, NT_{driver}, NT_{policyholder}, NT_{date}, NT_{name}, NT_{id}, NT_{vehicle}, NT_{brand}, NT_{model}, NT_{registrationnumber}, NT_{project}, NT_{offers}, NT_{offer}, NT_{total}, NT_{amountofinstalments}, NT_{numberofinstalments}\}$

$\sum = \{Element_{fare}, Element_{owner}, Element_{driver}, Element_{policyholder}, Element_{date}, Element_{name}, Element_{id}, Element_{vehicle}, Element_{brand}, Element_{model}, Element_{registrationnumber}, Element_{project}, Element_{offers}, Element_{offer}, Element_{total}, Element_{amountofinstalments}, Element_{numberofinstalments}, Literal\ \}$

R={

r1: NT_{fare}::= $Element_{fare}$ (NTvehicle NT_{owner} NT_{driver} $NT_{policyholder}$ NT_{date})

r2: NT_{owner}::= $Element_{owner}$ (NT_{name} NT_{id})

r3: NT_{driver} ::= $Element_{driver}$ (NT_{name} NT_{id})

r4: $NT_{policyholder}$ ::= $Element_{policyholder}$ (NT_{name} NT_{id})

r5: NT_{date} ::= $Element_{date}$ (Literal)

r6: NT_{name} ::= $Element_{name}$ (Literal)

r7: NT_{id} ::= $Element_{id}$ (Literal)

r8: $NT_{vehicle}$ ::= $Element_{vehicle}$ (NT_{brand} NT_{model} $NT_{registrationnumber}$)

r9: NT_{brand} ::= $Element_{brand}$ (Literal)

r10: NT_{model} ::= $Element_{model}$ (Literal)

r11: $NT_{registrationnumber}$ ::= $Element_{registrationnumber}$ (Literal)

r12: $NT_{project}$ ::= $Element_{project}$ (NT_{date} NT_{offers})

r13: NT_{offers} ::= $Element_{offers}$ (NT_{offer})

r14: NT_{offer} ::= $Element_{offer}$ (NT_{total} $NT_{amountofinstalments}$ $NT_{numberofinstalments}$)

r15: NT_{total} ::= $Element_{total}$ (Literal)

r16: $NT_{amountofinstalments}$ ::= $Element_{amountofinstalments}$ (Literal)

r17: $NT_{numberofinstalments}$ ::= $Element_{numberofinstalments}$ (Literal) }

Figure 5: Example of RTG grammar.

The lowering process starts from the initial symbol S and follows the production rules of the grammar G to create a valid XML Document T(∑). It is driven by the mappings M that decide how production rules must to be applied.

Starting from node v, mappings determine movements in the graph D_{rq}. The pseudocode of the lowering function is presented in Algorithm 1. A production rule can be applied multiple times to produce repeated structures, for instance, a list of elements. Ambiguity may arise, however, during the execution of the lowering. If the same non-terminal symbol has been mapped to different paths, and more than one of these paths can be simultaneously satisfied in the graph, then a fatal error is raised. All the error situations are described in more detail below:

- Line 5, the derivation tree reaches a terminal symbol of type $Literal_{value}$. However, the current node in the graph is not an RDFLiteral, therefore it is not possible to determine the value for the terminal symbol.
- Line 15, the algorithm finds a non-terminal symbol NT_i and it is not possible determine the production rule to be applied. This error is impossible if the grammar is built as a previous work [Rodríguez et al. 2009] introduces.
- Line 25, the algorithm is processing a non-terminal symbol NT_i with several mappings M^{NTi} and none of them can be applied in the current graph D_{rq}. Note that this is not the same as a non-terminal symbol which does not have any mappings. In the latter case, the algorithm continues without errors.
- Line 27, it is the opposite case to the previous one. More than one mapping rule for a non-terminal symbol NT_i can be simultaneously applied taking into account the current graph D_{rq}. The algorithm stops because there is no way to determine which one is preferred.

3.4 L_{ifting} function

Given a regular tree grammar, G, a syntactic response comprised by a tree of terminals T(∑) and a set of mappings M, we de ne the lifting process as the transformation function of the syntactic information to a semantic representation Drp:

$$L_{ifting} : T(\sum) \times G \times M \rightarrow D_{rp}$$

The lifting process is realized by parsing T(∑). For each terminal symbol, its mapping rule is located and a subgraph is created by instantiation of the graph path of the mapping rule. The subgraphs are merged recursively. The pseudocode for the lifting function is presented in Algorithm 2. This algorithm uses the following auxiliary functions:

- FreshBlankNode: produces a new blank node in the graph.
- GetNonTerminal: takes a terminal symbol and a context. It returns the nonterminal symbol in the left side of the production rule that generates terminal symbol in that context. In other words, let R0 be the subset of the production rules that have been applied so far in the derivation tree from the initial symbol to the current symbol, then: GetNonTerminal(Elementi,ctx) = {NTi ϵ NT/(NTi ::= Elementi(α)) ϵ R'} (1)
- MergeGraphMergingNode: merges two graphs, and additionally unifies two nodes.
- Get-/SetGraphRoot: in general, graphs do not have a root node. However, the algorithm needs to temporarily mark a node as "root" to drive the merging of the subgraphs. This pair of functions gets and set the root node of a graph.
- InstantiateGraphPath: given an initial node, a path and an original graph, this function creates a new graph that contains the original graph and a new path across a sequence of new blank nodes from the initial node to the root node of the original graph [see Fig. 6]. In the new graph the root node is the initial node.

Algorithm 1 Lowering

```
Require: G, symbol, ctx, M, $D_{rq}$, vertex
Ensure:                T(∑)                ≠                ∅
if symbol is Literal then
 1:    if vertex is $RDFLiteral_{value}$ then
 2:       return $Literal_{value}$
 3:    else
 4:       Error("Vertex isn't RDF Literal")
 5:    end if
 6: else if symbol is $Element_{QName}$ then
 7:    for all child ϵ symbol.children do
 8:    childrenResult ← Lowering(G, child, ctx, M, $D_{rq}$, vertex)
 9:    end for
10:    return $Element_{QName}$(childrenResult)
11: else if symbol ϵ NT then
12:    r: symbol → $Element_{QName}$ ϵ R
13:    if ∃! r then
14:       Error("Symbol doesn't have production rule")
15:    else
16:       if $M^{symbol} = \emptyset$ then
17:          for all child ϵ $Element_{QName}$:children do
19:             childrenResult ← Lowering(G, child, ctx + symbol, M, $D_{rq}$, vertex)
20:          end for
21:          return childrenResult
22:       else
23:          $M_{valid}$ = {m ϵ $M^{symbol}$/m:(ctx, symbol, α), α is satisfiable for vertex}
24:          if $M_{valid} = \emptyset$ then
25:             Error("Symbol doesn't have valid mapping")
26:          else if $|M_{valid}| > 1$ then
27:             Error("Symbol has more than one valid mapping")
28:          else
29:             for all v ϵ $[\alpha]^{D_{rq}}$ (vertex) do
30:                for all child ϵ $Element_{QName}$.children do
31:                   childrenResult ← Lowering(G, child, ctx + symbol, M, $D_{rq}$, v)
32:                end for
33:             end for
34:             return childrenResult
35:          end if
36:       end if
37:    end if
38: else
39:    Error("Unknown symbol")
40: end if
```

```
Algorithm 2 Lifting
Require: G, symbol, ctx, M, T(∑)
Ensure: $D_{rs} \neq \emptyset$
 1: if symbol is $Literal_{value}$ then
 2:     return $D_{rs} \leftarrow (RDFLiteral_{value}, \emptyset)$
 3: else if symbol is $Element_{QName}$ then
 4:     for all child ϵ symbol.children do
 5:         childsResult ← Lifting(G,child, ctx + nonterminal, M, T(∑))
 6:         if child 2 Literal then
 7:             return childsResult
 8:         else if child ϵ $Element_{QName}$ then
 9:             $D_{rs} \leftarrow (\emptyset,\emptyset)$
10:             blankNode ← FreshBlankNode($D_{rs}$)
11:             nonTerminal ← GetNonTerminal(symbol, ctx)
12:             nonTerminalChild ← GetNonTerminal(child, ctx + nonTerminal)
13:             if $M^{nonTerminalChild} = \emptyset$ then
14:                 return MergeGraphMergingNode(blankNode, $D_{rs}$, childResult)
15:             else if $|M^{nonTerminalChild}| = 1$ then
16:                 m:(ctx, nonTerminalChild,α) ϵ $M_{nonTerminalChild}$
17:                 $D_{rs}$ ← InstantiateGraphPath(blankNode, α ,childResult)
18:             else
19:                 Error("Symbol has more than one mapping")
20:             end if
21:             SetGraphRoot($D_{rs}$, blankNode)
22:             return $D_{rs}$
23:         end if
24:     end for
25: end if
```

Similarly to the lowering process, the lifting process may run into ambiguous cases. If a non-terminal symbol has several mapping rules and they are simultaneously valid in a certain context, then it is not possible to determine which path must be instantiated, and consequently an error is raised (Line 19).

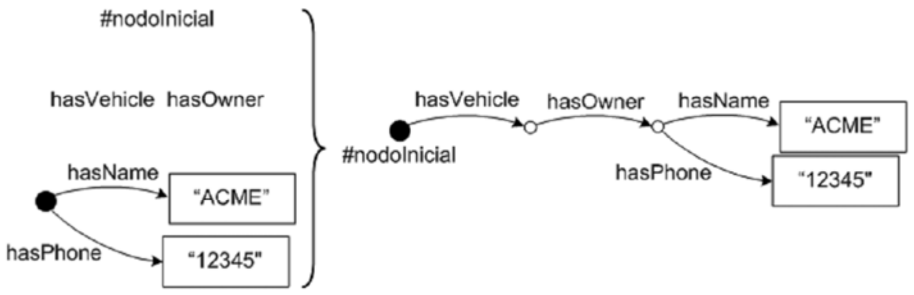


Figure 6: The function InstantiateGraphPath produces a new graph that is a superset of the original one, with the addition of a new path.

```
@prefix onto: <http://example.org/ontology#>.
onto:profile1 onto:hasPolicyHolder onto:companyOne .
onto:profile1 onto:hasDriver onto:personOne .
onto:profile1 onto:hasVehicle onto:car .
onto:profile1 onto:hasDate "10/05/2008" .
onto:car onto:hasBrand "Renault" .
onto:car onto:hasModel "Megane" .
onto:car onto:hasRegistrationNumber "1234AAA" .
onto:car onto:hasOwner onto:companyOne .
onto:personOne onto:hasName "Scott" .
onto:personOne onto:hasNaturalPersonsRegister "98765432R" .
onto:companyOne onto:hasName "ACME" .
onto:companyOne onto:hasTaxIdentificationNumber "12345678N" .
```

Figure 7: Some facts described as triples in N3 syntax.

3.5 Execution traces

In order to illustrate the grounding functions with a trace, a graphical notation is introduced [see Fig. 8]. Mapping rules [see Fig. 9] are depicted as boxes, with the context and the non-terminal symbol in the left side, and the graph path in the right side.

3.5.1 Lowering trace

The notation for the states of the lowering function contains the current non-terminal symbol (upper half of the box) and the current node of the graph.

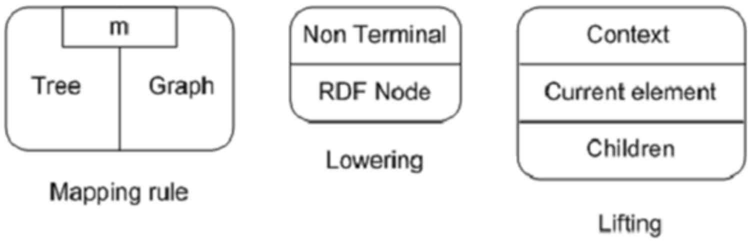


Figure 8: Graphical notation used to depict mapping rules and the states during the execution of the lowering and lifting functions, respectively

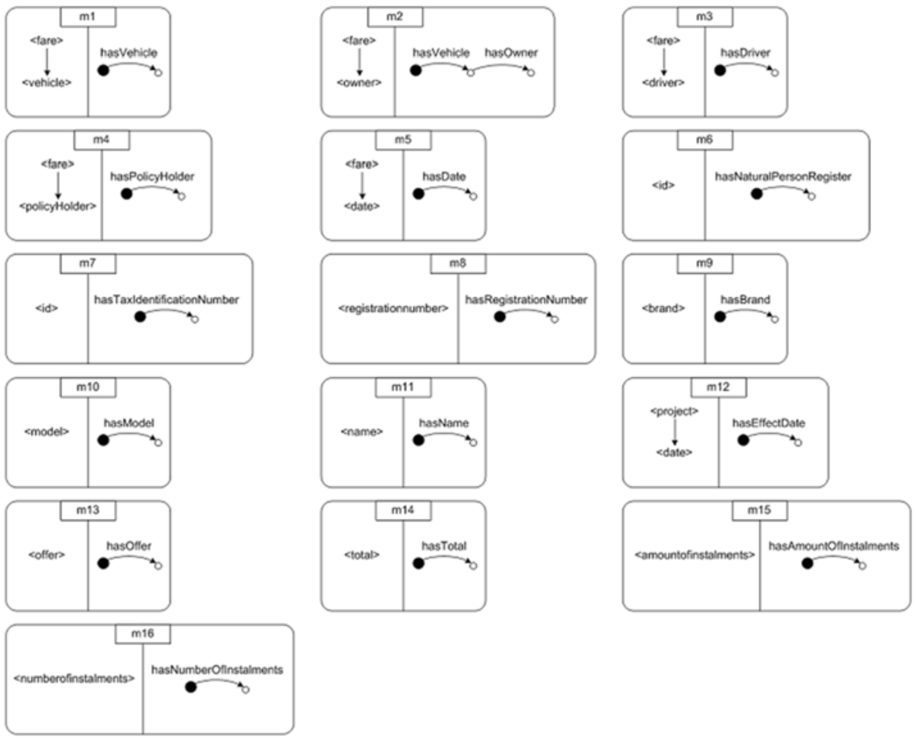


Figure 9: Mapping rules the graphical notation

Each state transition is annotated with the name of the production rule (r_i) and, optionally, the mapping rule (m_i). Figure 10 illustrates the complete trace of the lowering function using the following input parameters: 1) G = grammar [see Fig. 5]; 2) M = mappings [see Fig. 9]; 3) D_{rq} = input semantic data [see Fig. 7] and 4) v = profile1.

The initial state is the root of the derivation tree. Consequently, the non-terminal symbol is the initial symbol of the grammar, NT_{fare}. The first state transition is described in detail [see Fig. 11], and it is red by the production rule r1. The terminal symbol (fare) and its children are derived by the production rule. However, there is not any valid mapping rule for the current non-terminal symbol, therefore none is applied, and the active node in the graph is preserved in the children states. The process continues visiting the newly created children in the derivation tree. In the next step of the trace [see Fig. 12], the current non-terminal symbol is $NT_{vehicle}$. Production rule r8 is red. In this case, mapping rule m1 is valid; therefore, the new children states have a different active node, which is the result of traversing the edge hasVehicle in the graph from the previously active node. For convenience, the relevant portion of the D_{rq} graph is shown in the figure, as well as the valid mapping rule.

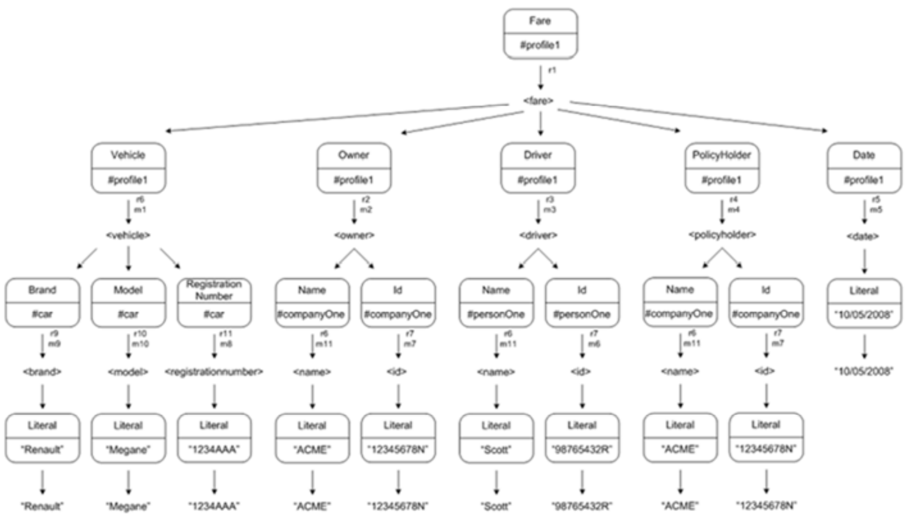


Figure 10: Execution of lowering, from the initial state containing the initial symbol (root of the derivation tree)

The process continues expanding all the non-terminal symbols created by execution of the production rules of the grammar. Eventually, the derivation tree reaches a leaf, i.e., a Literal terminal symbol. In this situation, the algorithm must assign the lexical value of the literal symbol. Assuming that the mapping rules are correctly built, at such point, the current node in the graph must be an RDFLiteral, which value is simply copied into the tree leaf. The same non-terminal symbol can appear in more than one mapping rule. In some of those situations, the context decides which mapping rules can be applied [see Fig. 13]. At this point, the non-terminal symbol is NT_{date}, which appears in two candidate mapping rules (m5 and m12). However, the context in the derivation tree (NT_{fare}) only matches with m5 and excludes m12. Consequently, this case is unambiguous.

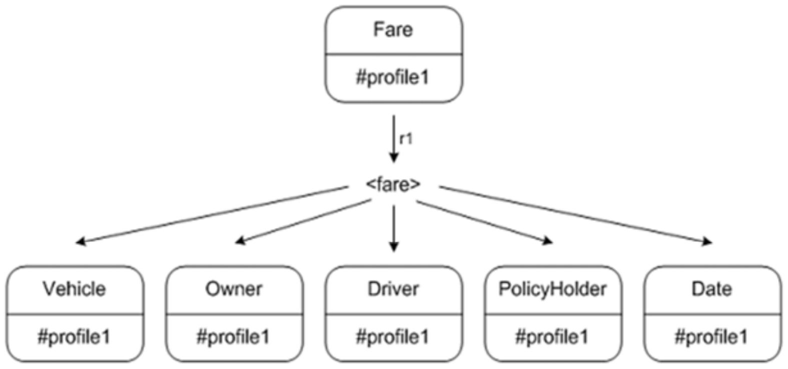


Figure 11: First state transition in the execution of the lowering function

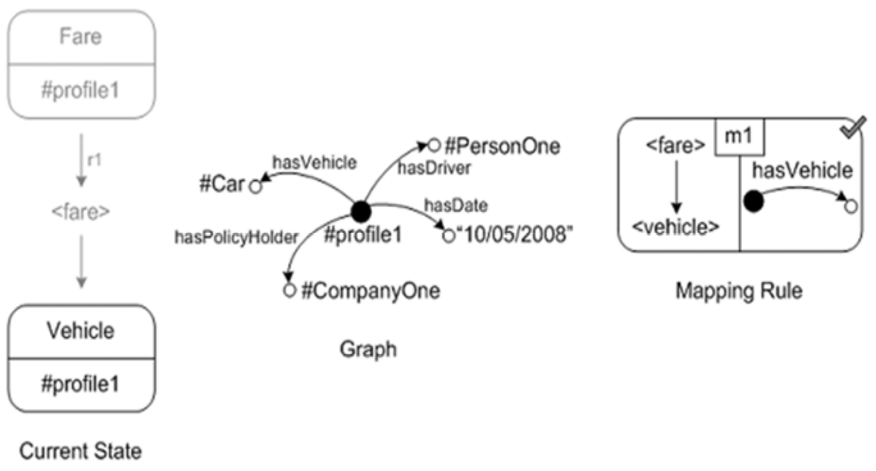


Figure 12: Second step in the execution of the lowering function

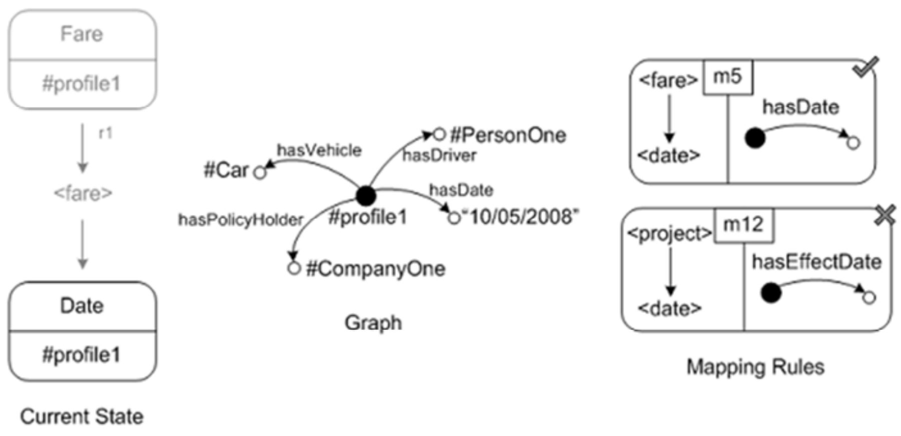


Figure 13: Situation in which the context decides which mapping rule is to be applied during the lowering process

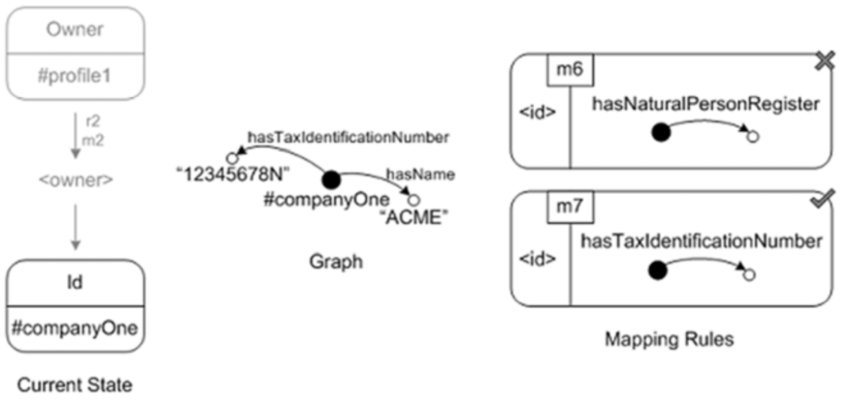


Figure 14: Multiple candidate mapping rules at this point. The path of the mapping rules determines which one is applied

Finally, there are some cases where even the context is insufficient. At the point depicted in Fig. 14, the non-terminal symbol NT_{id} is being processed. In this case there are two candidate mapping rules (m6 and m7), and they do not put any constraint on the context, therefore both are equally valid. In such case, the algorithm fallbacks to check whether the path of the mapping rules can be traversed from the current node in the graph. In the figure, from the node companyOne, only the path of m7 can be traversed, while there is not any edge matching the path of m6. Therefore, m7 is applied to move the active node to the literal node "12345678N". The final result is the tree composed of the terminal symbols, i.e., the result of removing the state boxes from Fig. 10. The result is converted in the XML Document [see Fig. 15].

```
<?xml version="1.0" encoding="utf-8" ?>
<fare>
 <vehicle>
   <brand>Renault</brand>
   <model>Megane</model>
   <registrationnumber>1234AAA</registrationnumber>
 </vehicle>
 <owner>
   <name>ACME</name>
   <id>12345678N</id>
 </owner>
 <driver>
   <name>Scott</name>
   <id>98765432R</id>
 </driver>
 <policyholder>
   <name>ACME</name>
   <id>12345678N</id>
 </policyholder>
 <date>10/05/2008</date>
</fare>
```

Figure 15: Example of XML Document.

3.5.2 Lifting trace

Each state of the recursive execution of the lifting function is depicted as a box [see Fig. 8] that contains the current terminal symbol (in the middle of the box), as well as its ancestor and child elements. Some state transitions re mapping rules that produce graph fragments. Figure 17 illustrates the complete trace of the lifting function for the following input parameters: 1) G = grammar [see Fig. 5]; 2) M = mappings [see Fig. 9] and 3) T(∑) = tree of the terminal symbols in the XML Document [see Fig. 17].

The process starts by parsing the terminal symbol at the root of the XML tree, in this case, $\text{Element}_{\text{project}}$. According to the grammar, this terminal symbol is derived from the non-terminal symbol $\text{NT}_{\text{project}}$ by production rule r12. As there is not any mapping rule valid for this non-terminal symbol, the parsing process simply continues recursively by the child branches, without producing any graph. Later, the terminal symbol $\text{Element}_{\text{date}}$ is found by the parser [see Fig. 16]. It is derived from the non-terminal symbol NT_{date} by production rule r5. There are two mapping rules (m5 and m12) available for this non-terminal symbol, but only m12 matches the context $\text{NT}_{\text{project}}$, therefore it is red.

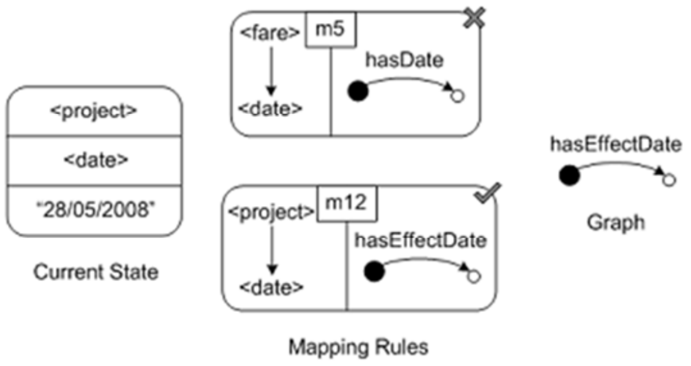


Figure 16: Sample situation in which the context of mapping rules decides which one is to be applied during lifting

A new subgraph is created by instantiating the path described in the mapping rule. The process continues descending to the child elements. Eventually, the function reaches a leaf, for instance, the terminal symbol $\text{Literal}_{25=08=2008}$. This is a final case; a minimal graph is created with just an RDFLiteral node with the lexical value of the terminal symbol. Finally, the recursive calls return, and the postprocessing stage merges the subgraphs produced by the child branches (root nodes are used as merging points), completing the final graph [see Fig. 18, bottom].

```
<?xml version="1.0" encoding="utf-8" ?><project>
<date>28/05/2008</date> <offers> <offer> <total>1200</total>
<amountofinstalments>100</amountofinstalments>
<numberofinstalments>12</numberofinstalments> </offer> ... <offer>
<total>1200</total> <amountofinstalments>600</amountofinstalments>
<numberofinstalments>2</numberofinstalments> </offer>
</offers></project>
```

Figure 17: Sample XML response obtained from a web service (without the SOAP envelope)

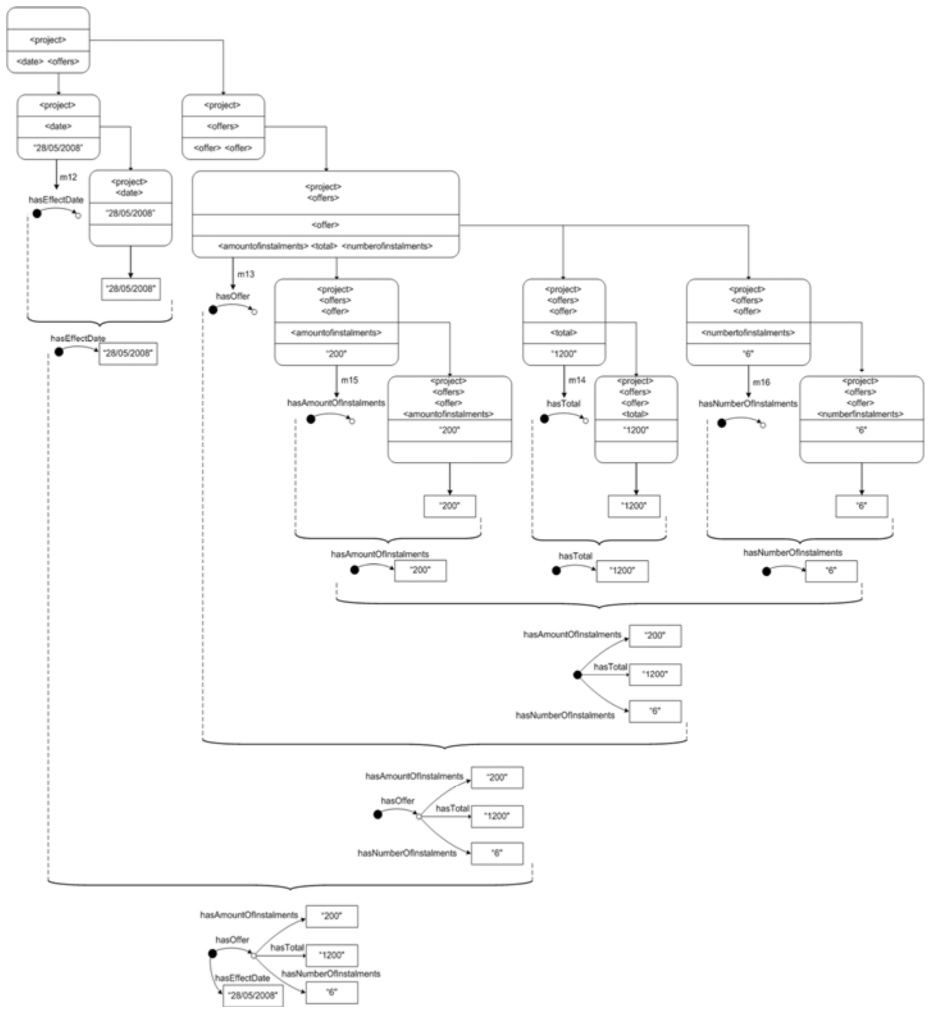


Figure 18: Execution of lifting, from the XML tree (top part of the figure) to the resulting RDF graph (bottom of the figure)

4 Experimentation: the PRAVIA project

E2000 Nuevas Tecnologías is the leading company in the insurance broker sector in Spain. The PRAVIA project was launched by E2000 [http://www.e2000.es] in partnership with Fundación CTIC and its main objective was to test the applicability of semantic web services to this business domain. For the sake of simplicity, only two business processes and two service providers were chosen. This sample has proven enough to create a realistic environment with non-trivial web services.

Feature	Our approach	SAWSDL	Ad-hoc
Use of XML Schema annotations	+, xsd:annotations	-, new attributes	-
Programming skills required	-	+	+
Complete solution	+	Runtime semantics undefined	+
Usable by domain experts	+	+	-
Easy Maintenance and Evolution (changes in ontologies or services)	+	+	-
Support for REST Web Services	-	-	+
Able to perform data mediation	-	+	+
Dependence on modeling language	+	+	+

Table 1: Comparative study of grounding approaches

The Web Service Execution Environment (WSMX) platform, the WSMO ontology and the WSML language were selected to deploy the semantic web services technology in this controlled environment. The grounding process provided by the WSMX Communication Manager was found to be in an early stage of development, and it could not address the requirements of the business environment. Therefore, it was replaced with an extended component with the same interface. The new component uses the mapping language and implements the transformation functions described in this paper.

Our approach to data grounding in semantic web services takes into account just the data structure definition (using XML Schema) of the whole WSDL description of a web service. In order to introduce the mapping rules in the XML Schema, we use the elements xsd:annnotation and xsd:appinfo and its at-tribute source. Each annotation is interpreted as one or more mapping rules. The non-terminal symbol of the mapping rule is the symbol labelled after the xsd:element. The context is obtained as the list of non-terminal symbols be-tween the current xsd:element declaration and the opening declaration of an XML Schema type. The content of an annotation in the source attribute is a graph path, described by a list of IRIs separated by blank spaces. By using xsd:appinfo, it is possible to introduce multiple mappings associated to the same element. According to Tab. 1 our proposal is compared to other approaches, we do not evaluate the details of the implementation of each solution, and we only focus on some general properties. The most relevant feature of our approach is its complete declarativeness (it does not require any programming skill), and it improves the maintenance and evolution of the knowledge bases as well as web services. The main drawback is the limited support for proto-cols (it is only supported WSDL 1.1 and SOAP 1.x) but it can be extended to do and process annotations inside the Web Application Description Language (WADL), a machine-readable description of HTTP-based web applications, basically REST-oriented services.

5 Conclusions and future work

Our proposal for data grounding relies on a mapping language that describes relations as mapping rules and a processor to execute the rules and make trans-formations between the syntactic description of the messages of a web service and the semantic model. We pursue a domain-independent and systematic solution that reduces to the minimum the human intervention. Following, main remarks about this approach for data grounding are presented: 1) it require human intervention on design time. The difference lies in which tasks are required and who can do them. Our solution only needs some simple work to map syntactic descriptions to semantic entities. This operation can be carried out using a simple graphical user interface; 2) it depends on the web service and se-mantic technologies. We only support WSDL (document based) 1.1 and SOAP 1.x. On the contrary, it is independent from the ontology language, existing web standards for graph-based knowledge representation systems are supported; 3) it positively contributes to the maintainability, reliability, robustness, time to repair and cost of the SOA architectures in the new Cloud Computing realm, as a consequence of removing the need of ad-hoc developments. This is a distinctive feature of our approach with respect to manually implementing data grounding using languages such as XSLT, XSPARQL or Java; 4) it must not be confused with data mediation. Although some shared transformation patterns can be identified, data mediation and data grounding are conceptually different, and their responsibilities should be clearly separated in a successful semantic web services platform; 5) it is a partial solution for semantic web services grounding. Data grounding is able to build requests and to process responses, but that is just a part of SWS grounding. There are other issues that must be addressed separately (e.g., authentication) and 6) it is a viable solution for data grounding of semantic web services. Without data grounding, SWS platforms cannot invoke real web services. However, we cannot assure that all SOAP-based web services will be compatible with our solution because heterogeneities are always present due to tools, extensions in specifications, etc.

We have reused the experience of previous works and specifications [Martin et al. 2004, Farrell and Lausen 2007, de Bruijn et al. 2005, Akkiraju et al. 2005] in the semantic web services area. Our solution was embedded into WSMX to put it to test in a real environment (web services coming from the insurance sector). The results con rm that it is possible to specify data grounding of web services using a mapping language. Anyway, real web services are not limited to those accepting SOAP messages and described by WSDL. Therefore, we are working to extend our solution to support other kinds of service (message protocol and description format), such as REST services, WSDL 2.0, etc. We also study the alignment of our solution with other proposals and recommendations from W3C and OASIS. Finally, taking into account the new semantic features of HTML5 [http://dev.w3.org/html5/spec/single-page.html] and its implicit XML grammar, this solution could be applied to make transformations (similar to XSL) between different datasources (web services, databases, etc.) to finally get a HTML5 representation keeping all semantics from the source.

References

[Akkiraju et al. 2005] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T.: "Web Service Semantics-WSDL-S"; W3C member submission; W3C (2005); http://www.w3.org/Submission/WSDL-S/.

[Álvarez Sabucedo and Rifón 2010] Álvarez Sabucedo, L., Rifón, L. A.: "Locating and Crawling eGovernment Services A Light-weight Semantic Approach"; Journal of Universal Computer Science; 16 (2010), 8, 1117--1137.

[Anicic et al. 2006] Anicic, D., Brodie, M., Bruijn, J. D., Fensel, D., Heymans, S., Ho mann, J., Kerrigan, M., Kopecky, J., Krummenacher, R., Lausen, H., Mocan, A., Toma, I., Zaremba, M.: "Semantically enabled service oriented ar-chitectures: A manifesto and a paradigm shift in computer science"; Technical report; In Proceedings of WICI International Workshop on Web Intelligence (WI) meets Brain Informatics (BI) (WImBI 2006) (2006).

[Balzer and Liebig] Balzer, S., Liebig, T.: "Bridging the Gap Between Abstract and Concrete Services"; A Semantic Approach for Grounding OWL-S; Proceedings of the Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications (2004).

[Battle and Bernstein 2005] Battle, S., Bernstein, A.: "Semantic Web Services Framework (SWSF)"; W3C member submission; W3C (2005); http://www.w3.org/Submission/SWSF/.

[Bohring and Auer 2005] Bohring, H., Auer, S.: "Mapping XML to OWL Ontologies."; K. P. Jantke, K.-P. F•ahnrich, W. S. Wittig, eds., Leipziger Informatik-Tage; volume 72 of LNI; 147--156; GI, 2005.

[Booth et al. 2004] Booth, D., Haas, H., McCabe, F., (until October 2003), E. N., (until March 2003), M. C., (until March 2003), C. F., (until March 2003), D. O.: "Web Services Architecture"; W3C working group note; W3C (2004); http://www.w3.org/TR/ws-arch#introduction.

[Bruijn et al. 2006] Bruijn, J. d., Ehrig, M., Feier, C., Mart n-Recuerda, F., Schar e, F., Weiten, M.: "Ontology mediation, merging and aligning"; Semantic Web Technologies; Wiley, 2006.

[Burstein et al. 2005] Burstein, M. H., Bussler, C., Zaremba, M., Finin, T. W., Huhns, M. N., Paolucci, M., Sheth, A. P., Williams, S. K.: "A Semantic Web Services Architecture."; IEEE Internet Computing; 9 (2005), 5, 72--81.

[Cabral et al. 2004] Cabral, L., Domingue, J., Motta, E., Payne, T. R., Hakim-pour, F.: "Approaches to Semantic Web Services: an Overview and Comparisons"; ESWS; 225--239; 2004.

[Calladine and Downey 2005] Calladine, J., Downey, P.: "Xml Schema and Web Services"; W3C Workshops on XML Schema 1.0 User Experiences; 2005.

[Comon et al. 2007] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacque-mard, F., Lugiez, D., Tison, S., Tommasi, M.: "Tree Automata Techniques and Applications"; (2007); release October, 12th 2007.

[Corradini et al. 1996] Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Loew, M.: "Algebraic Approaches to Graph Transformation, part I: Basic Concepts and Double Pushout Approach"; Tr-96-17; Universitá di Pisa, Dipartimento di Informatica (1996).

[Cowan and Tobin 2004] Cowan, J., Tobin, R.: "XML Information Set (Second Edition)"; W3C recommendation; W3C (2004); http://www.w3.org/TR/xml-infoset/.

[de Bruijn et al. 2005] de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., et al.: "Web Service Modeling Ontology (WSMO)"; W3C member submission; W3C (2005); http://www.w3.org/Submission/WSMO/.

[Fallside and Walmsley 2004] Fallside, D. C., Walmsley, P.: "XML Schema Part 0: Primer Second Edition"; W3C recommendation; W3C (2004); http://www.w3.org/TR/xmlschema-0/.

[Farrell and Lausen 2007] Farrell, J., Lausen, H.: "Semantic Annotations for WSDL and XML Schema"; W3C recommendation; W3C (2007); http://www.w3.org/TR/sawsdl/.

[Fernández et al. 2010] Fernández, S., Berrueta, D., Rodríguez, M. G., Gayo, J. E. L.: "Trioo - Keeping the Semantics of Data Safe and Sound into Object-oriented Software"; ICSOFT (1); 311--320; 2010.

[García et al. 2011] García, J., Peñalvo, F. J. G., Theron, R., de Pablos, P. O.: "Usability Evaluation of a Visual Modelling Tool for OWL Ontologies"; J. UCS; 17 (2011), 9, 1299--1313.

[Klímek and Necaský 2011] Klímek, J., Necaský, M.: "Generating Lowering and Lifting Schema Mappings for Semantic Web Services"; Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications; WAINA '11; 29--34; IEEE Computer Society, Washington, DC, USA, 2011.

[Kopecký 2005] Kopecký, J.: "Simple RDF to XML Data Grounding (slides)" (2005).

[Kopecký et al. 2007] Kopecký, J., Moran, M., Vitvar, T., Roman, D., Mocan, A., eds.: "D24.2v0.1. WSMO Grounding"; WSMO, 2007.

[Kopecký et al. 2006] Kopecký, J., Roman, D., Moran, M., Fensel, D.: "Semantic Web Services Grounding"; AICT/ICIW; 127; IEEE Computer Society, 2006.

[Kopecký and Schütz 2008] Kopecký, J., Schütz, A.: "D1.2.1 WSMO grounding in SAWSDL"; SOA4All deliverable d1.2.1; European Project-FP7 (2008); http://www.soa4all.eu.

[Martin et al. 2004] Martin, D., Burstein, M., et al.: "OWL-S: Semantic Markup for Web Services"; W3C member submission; W3C (2004); http://www.w3.org/Submission/OWL-S/.

[Mocan and Cimpian 2005] Mocan, A., Cimpian, E., eds.: D13.3v0.2 WSMX Data Mediation; WSMO, 2005.

[Murata et al. 2001] Murata, M., Lee, D., Mani, M.: "Taxonomy of XML Schema Languages using Formal Language Theory"; Extreme Markup Lan-guages; Montreal, Canada, 2001.

[Pan et al. 2011] Pan, Y., Tang, Y., Li, S.: "Web Services Discovery in a Pay-As-You-Go Fashion"; Journal of Universal Computer Science; 17 (2011), 14, 2029{2047.

[Pantschenko et al. 2005] Pantschenko, K., Noppens, O., Liebig, T.: "Grounding Web Services Semantically: Why and How?"; W3C Workshop on Frameworks for Semantics in Web Services; 2005.

[Pedrinaci and Domingue 2010] Pedrinaci, C., Domingue, J.: "Toward the Next Wave of Services: Linked Services for the Web of Data"; Journal of Universal Computer Science; 16 (2010), 13, 1694--1719.

[Rodríguez et al. 2009] Rodríguez, M. G., Alvarez, J. M., Berrueta, D., Polo, L.: "Declarative Data Grounding Using a Mapping Language"; Communications of SIWN; 1 (2009), 7.

[Roman et al. 2006] Roman, D., de Brujin, J., Mocan, A., Toma, I., Lausen, H., Kopecky, J., Bussler, C., Fensel, D., Domingue, J., Galizia, S., Cabral, L.: "Semantic Web Technologies". Trends and research in ontology-based systems; Wiley, 2006.

[Taniar et al. 2011] Taniar, D., Khalil, I., Pardede, E.: "Cloud Computing"; Journal of Universal Computer Science; 17 (2011), 8, 1134—1134.