Ontology-enhanced description of traceability services
Viorica R. Chifu, Ioan Salomie, Emil Şt. Chifu
Department of Computer Science, Technical University of Cluj-Napoca, Barițiu 28, Cluj-Napoca

Abstract

Web services are software components that were designed to improve interoperability and integration of applications developed on different platforms. Automatic Web service composition and dynamic service discovery and selection are the most promising features that can be provided by Web services. These features can be achieved by adding semantics to the different layers of the Web service conceptual stack. The semantic information is provided by ontologies. This paper proposes an ontology model which allows semantic annotation of Web services aiming at automatic Web service composition for food chain traceability. The proposed model consists of a core ontology and two categories of taxonomic trees: Business Service Description trees and Business Product Description trees.

The model has been implemented in the framework of the Food-Trace project for traceability in the domain of meat industry.

1. Introduction

“Web services are software components that are described via WSDL (Web Service Description Language) and are capable of being accessed via standard network protocols such as but not limited to SOAP over HTTP” [23]. Web services offered by multiple business partners allow their interconnection, for Business-to-Business interoperability (B2Bi). Given the dynamic environment of the business processes, it is desirable to find Web services on the fly when creating complex business processes. Key steps in achieving this capability are the automated discovery, composition and execution of Web services.

The standard technologies for Web services (WSDL and UDDI) do not provide the necessary semantic information for a system to do automatic service discovery, composition, and execution. This limitation can be solved by the semantic Web technology, which associates semantic descriptions to each Web service. A variety of semantic information covering input/output data, functionality, execution, and quality of service can be added to Web service descriptions. This kind of semantics is provided by ontologies. An ontology is a “formal specification of a shared conceptualization” [15] in a human understandable and machine readable form, which consists of concepts, relations and restrictions. Ontologies provide a common formal vocabulary used in the construction of semantic Web service descriptions.

This paper proposes an ontology model which allows semantic annotation of Web services aiming at automatic Web service composition for food chain traceability. Our model consists of a core ontology and two categories of taxonomic trees: Business Service Description (BSD) trees and Business Product Description (BPD) trees. The core ontology expresses the basic concepts used for the semantic annotation of Web services. These concepts are common across a variety of food industry business processes and provide the basis for further specialization. The ontology model can be easily adapted to different though similar business domains by appending domain specific sub-trees of concepts under the appropriate nodes of the ontology model.

An ontology for the Romanian language has been built according to this model, to be used for traceability in the domain of food industry. The ontology describes the participants involved in the traceability chain, the services and products they offer/use, and the main features of the products. The product and feature concepts of the ontology are organized in taxonomic structures, which have been automatically built. The taxonomy learning is based on hierarchical self-organizing maps [6]. The candidates for concept names are collected by mining text corpora. The term extraction process is based on recognizing linguistic patterns in the text corpus.

The paper is organized as follows. Section 2 presents the ontology model and its design criteria, while section 3 details the implementation of the ontology model and presents a qualitative evaluation of the experimental results of the taxonomy learning process. Adding semantics to WSDL descriptions based on this ontology model is discussed in section 4.
1.1. Background

Commerce globalization is widening the distribution area of goods and services, making more difficult to control the problems of quality and consumer protection. To deal with competition, many food product companies have been constrained to adopt traceability systems along with the whole food chain. External traceability refers to systems aimed to allow the traceability of a product and/or attribute(s) of that product through the successive stages of the distribution chain, while the internal traceability is the traceability of raw materials, intermediate and final products within a productive or commercial unit [21].

The Food-Trace system represents a solution to assure the internal and external traceability in the domain of food industry, more exactly in meat processing industry. The system is developed as a service oriented architecture, based on the SOAP, XML, BPEL, UDDI, and WSDL standards to assure the interoperability at the levels of platform and programming language. The whole chain of external traceability is thought of as a Web service composition, where the constituent services provide some functionality offered by different participants in the food chain. The ontological component of the system helps to automate the composition process. The services that participate in the composition process are annotated with semantic information supplied by the ontology.

2. Ontology model and design criteria

The following design criteria have been considered for the ontology model:

- to be a simplified view of the business domain;
- to have a structure that can be easily adapted to other different thought similar business domains;
- to define taxonomies and well-structured relations in order to avoid redundancy and ensure inference accuracy;
- to provide the necessary semantic information in order to do automatic service discovery, selection and composition of Web services.

Based on these assumptions, we have developed an ontology model that serves as a common data model for describing service capabilities and for searching and selecting services. Moreover, this ontological data model also defines the vocabulary used in the automated planning of the service composition.

Our ontology model contains classes, properties and restrictions. The classes are concepts from the food traceability domain, which are organized in hierarchical structures. A property either defines a relation between concepts or a restriction. There are two types of relations in our ontology: hierarchical relations, and associative relations. The hierarchical relations build the taxonomic “is-A” and “part-of” structure of the ontology and associative relations relate concepts across the hierarchical structures. Associative relations have a domain corresponding to values from one class resulting in a range of values of another class.

This paper proposes an ontology model consists of a core ontology and two categories of taxonomic trees: Business Service Description trees and Business Product Description trees. Business Service Description trees represent all the concepts associated with specific services involved in the food chain traceability, while Business Product Description trees reflect specific concepts associated with the products provided by a business actor (product name, Universal Product Code, price, quantity). Our ontology model provides the semantic information necessary to describe each service according to its input, output and functionality. The proposed ontology model is used for adding semantics to WSDL and UDDI descriptions.

2.1. Semantic data model of the core ontology

The core ontology (Figure 1) defines six generic concepts: Business Actor, Service, Service Input, Service Output, Product and Feature.

![Figure 1: Core Ontology](image)

These concepts can be divided into two main domains: Business Service Descriptions (BSD) which contains concepts that semantically describe service inputs,
outputs and functionality and *Business Product Descriptions (BPD)* for concepts that qualify the inputs and outputs provided of a service. The relations between concepts are represented by connecting lines. The arrow connections represent functional relations (from domain to range), non-arrow connections represent one-to-many relations, and the diamond represents “part-of” aggregation relation. For example, *has-Input* is a functional relation from Service to Service Input, and *provideService* is a one-to-many relation from Business Actor to Service. Any Service Input or Service Output *has-part* Product and Feature entities.

### 3. Implementing the ontology model

The proposed model has been implemented in the framework of the Food-Trace project [20] for traceability in the domain of meat industry. The developed ontology is for the Romanian language. A view of the categories of our ontology is illustrated in Figure 2. The BSD part of the ontology has been developed in the Protégé ontology editor [13], while the BPD ontology part has been automatically built out of textual descriptions from Web sites of Romanian meat industry companies.

#### 3.1. Business Service Descriptions (BSD)

The BSD contains the following trees of concepts (taxonomies): Business Actor tree, Service tree, Service Input tree and Service Output tree. The Business Actor tree is a classification of the business actors involved in the food traceability. We have considered four generic classes of actors: Producers, Distributors, Transporters, and Customer Protection Organizations. Each of them features more specialized classes. For example, Producer is specialized as Food Producer, which is in its turn specialized as Salami and Sausages Producer and Dairy Producer.

The Service tree is a classification of the services provided by the business actors. As generic classes of services we have considered Order Service, Information Service and Customer Service, each of them featuring at their turn more specialized classes.

Finally, the Input tree and the Output tree are classifications of the inputs and outputs of the services respectively.

#### 3.2 Building Business Product Descriptions (BPD) using machine learning

The Business Product Description contains domain specific concepts which are organized in two taxonomic trees: Product and Feature. For our domain, Product tree is a classification of the meat products while Feature tree is a classification of features of these products (such as price, expiration date, temperature, quantity). The trees representing our domain taxonomy have been automatically built from a domain text corpus consisting of html pages with information about meat products. The pages were collected from Web sites of Romanian meat industry companies [18, 19]. The taxonomy learning process has two steps: term extraction step and taxonomy building and pruning step. In the term extraction step, the relevant terms (words or phrases) for the taxonomy building are extracted from the domain text corpus. The extracted terms become the candidates for the concept names in the final learnt taxonomy. In the taxonomy building and pruning step, the identified terms become concepts, and taxonomic (isA) relations are establish between them, by actually building a tree having the concepts in its nodes. The pruning phase eliminates the potentially uninteresting concepts from the taxonomy.

#### 3.2.1 Term extraction.

The candidates for concept names are identified in a three phase text mining process over the domain corpus. In the first phase a
Linguistic analysis

In the linguistic analysis phase, the domain text corpus is first annotated with information about the part of speech (POS) of every word with the help of the Brill POS tagger [2]. Brill tagger is a transformation-based rule tagger that is trainable on different languages. Since the entire ontology, including the domain taxonomy is for the Romanian language, the extracted terms are in Romanian, and the corpus is obviously completely written in the same language. Brill tagger can only be trained by a supervised learning process starting from an already POS tagged corpus. In order to train Brill tagger for Romanian, we used ROCO [16], an annotated Romanian text corpus which contains articles from Romanian newspapers.

Some corpus preprocessing was required for Brill tagger in order to be able to annotate our corpus [18, 19]. First, we have converted HTML documents to simple text files, then we have splitted all the documents in separate sentence. This preprocessed corpus is provided as input for the Brill tagger.

Our original (untagged) corpus consists of 130 documents collected from Web sites of the Romanian meat industry companies [18, 19]. Two experiments have been done with the Brill tagger. In the first one, we train the tagger on the whole ROCO corpus. Because the training time was too long, we decided to train the tagger only on part of the articles from the ROCO corpus (13 million of words). The evaluation of the trained tagger was performed on our corpus [18, 19]. In this case the accuracy calculated as the ratio of correct tags out of the total number of the tags, was 81%. For the second experiment, we split the (untagged) domain corpus into two corpora of equal size. The first one is labeled with part of speech tags after training the Brill tagger with the ROCO corpus. We then used this tagged corpus to train the Brill tagger for use on the second corpus. In this case the accuracy was significantly higher, 92%. The tagging accuracy is lower in the first case due to the lexical ambiguity of the words. The ROCO corpus and our corpus are taken from different domains and some words have different meanings depending on the context in which they appear. The corpus annotated in this way is then provided as input to a noun phrase chunker tool to identify domain concepts.

Identifying domain specific terms

The phase of identifying domain specific terms is based on recognizing linguistic patterns (noun phrases) in the domain text corpus. To extract domain specific terms from the corpus, we have implemented a noun phrase (NP) chunker which identifies noun phrases in the linguistically annotated text corpus. The chunker receives as input texts tagged with POS and provides as output tagged texts in which the identified noun phrases are annotated with a noun phrase tag. Our NP chunker is written by using lex and yacc. A context-free-grammar (CFG) to match simple noun phrases from Romanian natural language specifications was defined in yacc. Based on this CFG, a bottom-up parser is generated that uses shift-reduce parsing to recognize the noun phrases. The written yacc syntax rules of the grammar consist essentially of a head noun together with its pre/post-modifiers (attributes). The pre-modifiers of a head noun can be determiners, adjectives, adjectival phrases. The post-modifiers of the head noun can be possessive pronouns, adjectival phrase and prepositional phrases. In the Romanian language, like in the other languages, a noun phrase can be nested within another noun phrase, with no limit on the depth. This nesting process is represented in the grammar by recursive rules. Two kinds of recursive rules can be used to identify such language structures: direct recursive and indirect recursive rules. Our noun phrase chunker works well on the sublanguage of meat processing and product descriptions. For instance, consider the sentence: “Oferta de produse cuprinde aproximativ 65 de sortimente, punctul forte fiind reprezentat de specialitatile si produsele crud uscate.” (The product offer includes about 65 assortments, the strong point being represented by the specialties and the dry cruel products.) The chunker identifies “Oferta de produse”, “sortimente”, “punctul forte”, “specialitati”, and “produse crud uscate” as noun phrases.

Morphological analysis

Romanian language belongs to the Romance language family which also includes Italian, French, Spanish and Portuguese. In the Romanian language, nouns are inflected by gender (feminine, masculine and neuter), number (singular and plural) and case (nominative, accusative, dative, genitive and vocative). Adjectives and pronouns have the same gender, number and case with the noun they modify/refer to. Since the concepts of our taxonomy are designated by noun phrases we decided to do morphological analysis only for nouns, adjective and pronouns. By morphological analysis we reduce each inflectional word forms to its stem in canonical form. Our morphological analyzer is written in lex. We have defined regular expressions to identify the various inflectional word forms (nouns, adjective and pronouns). Starting from the assumption that each word form consist of a stem and its ending (according to the gender, number, case) some replacement rules.
have been defined for the Romanian language based on Romanian Academy Grammar [9]. For example, one of the rules applies to nouns which end in “uri” and states that the string “uri”, is remove in the canonical form of nouns. The agglutinative definite determiner for nouns in Romanian with its various flexional forms is also identified and removed from the affix (ending) of the nouns. The goal of the morphological analysis is to have a unique term originating from its various inflectional forms.

3.2.2 Taxonomy building and pruning. The taxonomy learning is based on hierarchical self-organizing maps, more specifically, on the Growing Hierarchical Self-Organizing Map (GHSOM) model [6]. In our setting, a learned GHSOM hierarchy is playing the role of a learned taxonomy. GHSOM is an extension of the Self-Organizing Map (SOM) learning architecture [11] - a popular unsupervised neural network model. The rectangular SOM map is a two-dimensional grid of neurons. Each input data item is classified into one of the neurons in the map. SOM clusters an input data space, giving rise to a similarity based smooth spread of the data items on the map. The data items must be represented as vectors of numerical attribute values.

The growing hierarchical self-organizing map model consists of a tree-like hierarchy of SOM’s [6]. The nodes in the tree are SOM’s that can grow horizontally during training by inserting either one more row or one more column of neurons. This happens iteratively until the average data deviation over the neurons in the SOM map decreases under a specified threshold $\tau_1$. The SOM’s of the nodes can also grow vertically during training, by giving rise to successor nodes. Each neuron in the SOM map is a candidate for expansion into a successor node. The expansion takes place whenever the data deviation on the current neuron is over a threshold $\tau_2$. The successor SOM map is then trained merely with the data subspace mapped into the parent neuron. The training of the whole GHSOM model converges and stops when both thresholds are satisfied. The depth and the branching factor of the hierarchy learned by GHSOM are controlled by the thresholds $\tau_1$ and $\tau_2$. The GHSOM learning behaves like a top-down process of hierarchical classification of the input data space items.

The noun phrases identified in the corpus are the terms in our setting, and these terms are classified in a GHSOM tree during the process of taxonomy building. To make possible the GHSOM classification of the terms, a vector representation for each term has to be chosen. In our setting, the attributes of the vector representation of a term encode the frequencies of occurrence for the term in different documents of the corpus.

Taxonomy pruning is achieved by avoiding terms occurring in too few documents of the corpus, specifically in less than 1-2% of the total number of documents in the corpus. Such terms cannot be considered as relevant to become concepts of the domain.

3.2.3 Experimental results. Below are some of the learned branches corresponding to the Product tree of the BPD trees. The English translations of the concepts of this taxonomy are given in italics. The concepts – nodes in the taxonomy – are represented as synonym sets, like in a thesaurus. The nodes represented by empty synonym sets are nodes with no concept label. They can actually be associated with a concept name by finding a common Romanian WordNet [17] hypernym of its successors [5].

```
{} { salam_turist_extra } extra tourist salami
{}      { salam_chorizo } Chorizo salami
{}         { salam_potoava, horseshoe salami
{}                        { salam_de vara_uscat } drying summer salami
{}                                      { salam_milano, salam_de porc,
{}                                                salam_canadian } Milano salami, pork salami
{}                                                     { salam_italian_extra, salam_sicilian,
{}                                                            salam_piept_pui_galinia,
{}                                                                salam_picant_extra, salam_taranesc,
{}                                                                      salam_sasesc_cu_verdeata,
{}                                                                          salam_sasesc_cu_ceapa, salam_palermo}
{}
{}      { salam_rustic, salam_cu sunca, salam_napoli,
{}                      salam_de vara_traditional, salam_de vara_extra }
{}                    { salam_ardelenesc }
{}                         { salam_victoria, salam_sasesc_cu_piper_verde }
{}
{} { salam_sinaia } Sinaia salami
{}      { salami }
{}
{}
{} Finally, below is a learned branch for the Feature tree.
{}
{} { compositia, aspectul exterior, composition
{}   tehnologie_de_obtinerer,
{}      conditie_de_pastrare, calitate_organoleptice }
{} { termen_de_valabilitate, expiration date
{}   recomandare_de_consum } consumption recommendation
{}
{}                     { condiment_naturale } natural spice
{}                                 { temperatura, umiditate } temperature, humidity
{}                                                      { sare } salt
{}                { zi } day
```

4. Semantic annotation of WSDL. Case study: Order Product service
In this section we illustrate how the ontology is used to annotate a Web service interface based on Semantic Annotations for WSDL and XML Schema specification (SAWSDL) [22]. SAWSDL defines how to add semantic annotations to various parts of a WSDL document such as input and output message structures, interfaces and operations. To accomplish semantic annotation, SAWSDL defines three types of extensibility attributes that can be applied to WSDL and XML Schema elements: ModelReference is used to annotate XML Schema complex type definitions, simple type definitions, element declarations, and attribute declarations as well as WSDL interfaces and operations; loweringSchemaMapping and liftingSchemaMapping, are added to XML Schema element declarations, complex type definitions and simple type definitions for specifying mappings between semantic data and XML. From all of these, only modelReference has been used in our work. The other two will be used in future developments in order to allow the handling of ontological heterogeneities between compatible concepts.

Our Product Order Web service expects a product name, a quantity, a delivery date and a unit of measure as service inputs. As a result of the service invocation the status of the order will be returned. In order to annotate the WSDL and XML Schema according to the SAWSDL specification, the following steps have been taken:

- add the namespace for the ontology and for SAWSDL to the definitions.
- annotate the WSDL operations with modelReference attributes pointing to the appropriate concept of the ontology.
- annotate the XML schema complex types corresponding to the input/output messages with modelReference attributes pointing to the appropriate concept of the ontology.

The WSDL and XML Schema including semantic annotations for this service is given in Figure 3. Each modelReference shown in Figure 3 identifies the concept in the ontology that describes the element to which it is attached. For instance, the “Produs” (product) element is described by the "ProdusDinCarne" (meat product) concept in the ontology. Our services are published by simply providing the UDDI registry with the URI for the SAWSDL files.

To understand how the semantic annotations are used for automating service discovery and composition we shortly describe our approach for automatic web service composition. In our approach the user selects his/her desired web services’ inputs and outputs (which are class concepts of the ontology, extracted by using Jena API) via a graphical user interface through a controlled language. From the GUI, the user invokes the Web Service Composer which generates an abstract solution. The Web Service Composer uses a planner which attempts to put together a chain of abstract services in order to generate a plan (abstract solution) that can satisfy the user requirements. The vocabulary used to express the initial state, the goal and the facts of the planner consists of concepts from the ontology. This abstract solution is provided to a Service Invoker which is responsible for the discovery of Web services in the UDDI registry. In order to parse the SAWSDL files, an extension of IBM WSDL4J library is used by the Service Invoker.

5. Related work

There is a considerable amount of research done in the ontology building domain. In this section, a couple of related ontology models and ontology learning frameworks are presented.

The WonderWeb project [14] was concerned with the development of an infrastructure for large-scale deployment of ontologies for the Semantic Web. The key concept of the infrastructure is represented by the ontologies describing the functionality of Semantic Web tools and services for RDF(S) storage and query. The main branches of such an ontology are Data (to describe the RDF(S) data structures) and Method (to describe the functionalities of the methods operating upon the data structures). Trying to make a comparison, the main branches of our ontology model are rather Products and Features (similar to Data) on one hand, and Services (similar to Method) on the other hand.

There is a multitude of ontology learning frameworks [7, 3]. We only enumerate two such frameworks as being the most related to ours.

In [1], the terms are represented with distributional (contextual) signatures, similar with our vectors of occurrences in different documents (contexts). The ontology learning approach is a top-down process, like the behaviour of our GHSOM based model. As opposed, the cited work uses decision tree learning, rather than neural learning.

A hierarchical self-organizing neural model is used in [10] to arrive at a taxonomy having concept labels only at the leaves. Concept names for the intermediate nodes of the taxonomy are found in a bottom-up process by querying WordNet for common hyponyms of brother nodes.

Our ontology learning is based on distributional similarity and clustering [3], where the clustering is
neural network driven. Most of the clustering based ontology learning approaches use the classical hierarchical clustering algorithm. The neural GHSOM model is better than the classical hierarchical clustering algorithm in terms of speed, noise tolerance and robustness [4].

6. Conclusions and future work

In this paper we presented a ontology model to facilitate the annotation and automated composition of Web services. The model consists of a core ontology and two categories of taxonomic trees: Business Service Description trees and Business Product Description trees. The proposed model was used to develop a business ontology for traceability in the domain of food industry. The domain specific concepts of this ontology are organized into a taxonomy which is automatically built out of textual descriptions from Web sites of Romanian meat industry companies. The experimental results obtained for the automatically built taxonomy are encouraging. Different approaches for automatic taxonomy building are hard to evaluate comparatively, since, even if the domain is the same, the authors use different corpora for their experiments. Moreover, our ontology is for the Romanian language, and we can not compare ourselves with other similar approaches from the same domain, because such results have not been reported, yet.

Our ontology is used for adding semantics to WSDL and descriptions, as vocabulary in the automated planning of the service composition and as vocabulary for ontological driven user interface.

In future work, we plan to implement a framework for automatic Web service composition based on our approach.

Acknowledgment

This work was supported by the Food Trace project within the framework of the “Research of Excellence” program initiated by the Romanian Ministry of Education and Research.

7. References

Table 3: A part of semantic annotation of XML Schema and WSDL for Order Product Web service

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:element name="IsAvailableResponse" type="tns:IsAvailableResponse"/>
<xs:complexType name="IsAvailable">
<xs:sequence>
<xs:element name="product" type="xs:string" minOccurs="0" sawsdl:modelReference="http://www.owl-ontologies.com/unnamed.owl#ProdusdinCarne"/>
<xs:element name="quantity" type="xs:int" sawsdl:modelReference="http://www.owl-ontologies.com/unnamed.owl#Cantitate"/>
<xs:element name="um" type="xs:string" minOccurs="0"/>
<xs:element name="date" type="xs:string" minOccurs="0" sawsdl:modelReference="http://www.owl-ontologies.com/unnamed.owl#Data"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="IsAvailableResponse">
<xs:sequence>
<xs:element name="return" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Order">
<xs:sequence>
<xs:element name="product" type="xs:string" minOccurs="0" sawsdl:modelReference="http://www.owl-ontologies.com/unnamed.owl#ProdusdinCarne"/>
<xs:element name="quantity" type="xs:int" sawsdl:modelReference="http://www.owl-ontologies.com/unnamed.owl#Cantitate"/>
<xs:element name="um" type="xs:string" minOccurs="0"/>
<xs:element name="date" type="xs:string" minOccurs="0" sawsdl:modelReference="http://www.owl-ontologies.com/unnamed.owl#Data"/>
</xs:sequence>
</xs:complexType>
.........................................................
.........................................................
```

```xml
<types>
<xsd:schema>
<xsd:import namespace="http://services/" schemaLocation="http://viorica-083908:1325/FoodTrace/Producator1Service/__container$publishing$subctx/WEB-INF/wsdl/Producator1Service_schema1.xsd"/>
</xsd:schema>
</types>
<message name="IsAvailable">
<part name="parameters" element="tns:IsAvailable"/>
</message>
<message name="IsAvailableResponse">
<part name="parameters" element="tns:IsAvailableResponse"/>
</message>
<message name="Order">
<part name="parameters" element="tns:Order"/>
</message>
<message name="OrderResponse">
<part name="parameters" element="tns:OrderResponse"/>
</message>
<portType name="Producator1">
<operation name="Order">
<input message="tns:Order"/>
<output message="tns:OrderResponse"/>
</operation>
</portType>
```