# AN AGENT-ORIENTED AND SERVICE-ORIENTED ARCHITECTURE IN MEDICINE

Florica MOLDOVEANU[1], Sorin-Alexandru CRISTESCU[2]

*Până de curând, sistemele informatice medicale erau proprietatea anumitor instituții, lucru care a avut un impact negativ asupra calității serviciilor medicale. Odată cu introducerea standardului HL7 v3 s-au creat premizele abordării problemei interoperabilității. Din păcate, folosirea standardului HL7 este limitată şi defectuos înțeleasă, în timp ce vechile aplicații încă domină lumea medicală. Această lucrare propune o arhitectură bazată pe sisteme multiagent şi web semantic care abordează problema interoperabilității între sistemele din domeniul medical, având ca nucleu un motor de căutare semantică pentru servicii web şi agenți software.*

*Traditionally, medical information systems have been proprietary to certain institutions, with a negative impact on the quality of medical services. The introduction of the HL7 standard version 3, helps in defining a better approach to the issue of interoperability. However, the usage of the HL7 standard is limited and misunderstood, while the traditional proprietary applications still dominate the medical world. This paper proposes an architecture based on multiagent systems and semantic web that addresses the interoperability between medical information systems, having as kernel a semantic web service as a solution to the registration and discovery of web services and software agents.*

**Keywords:** multiagent systems, semantic web, HL7, semantic search engine

## 1. Introduction

After the interoperability issue has been neglected for a long time, in the todays medical world there is a trend to develop and use standards that allow medical institutions or devices to exchange medical data. Such standards, like DICOM [1] (*Digital Imaging and Communications in Medicine*) and HL7 [2] (*Health Level 7*) are used by an increasing number of applications in various domains: patient monitoring, electronic patient record, exchange of medical images among devices produced by different vendors, etc.

[1] Prof., Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: fm@cs.pub.ro

[2] PhD student, Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: sorincalex@gmail.com

Moreover, HL7 version 3 has introduced real semantic interoperability through its new Reference Information Model (*RIM*). This opens up new possibilities in the way medical entities communicate.

This paper shows how the standards can be used in architecting a solution based on software agents and web services.

As an example we take as use case a solution for monitoring the state of health for elderly people. Such a person could wear a watch that functions as an embedded device on which a software agent is installed. The role of this agent is to monitor the patient's vital signals, such as blood pressure and pulse and inform an emergency service when necessary. For example if the patient faints without the possibility to alert the ambulance service, the agent would observe that the vital signs get beyond certain safety limits and thus communicate to an emergency service via wireless or 3G Internet.

The patient's personal agent would help in the same way in other similar circumstances, e.g. a car accident: as soon as the patient's vital signs deteriorate, the agent can contact the emergency service. Moreover, an agent embedded in the car could also take action, e.g. informing the police, the insurance company, etc. Agents make sense in such scenarios because of their autonomous behavior, i.e. they take actions based on observing the environment, without human intervention.

A service is basically an operation that obeys to a certain contract. Due to the heterogeneous character of the medical world, we can't expect to have the same services with the same input and output types being used by all medical institutions. The communication between medical institutions needs semantic interoperability, which can be obtained by employing *ontologies*. For example, when a certain hospital asks a service for the electronic patient record (EPR) of a certain patient, it needs to provide certain identification data for that patient and it needs to understand/interpret the received patient record. All pieces of information exchanged need to be expressed in ontologies understood by all parties involved. Likewise, when a doctor writes the diagnosis in the patient's medical record, the symptoms have to be mapped to certain ontology (e.g. SNOMED-CT [3] *(Systematized Nomenclature of Medicine -- Clinical Terms)*) such that they are understood by any other entity (human or not) which can interpret that ontology.

Web services have been around for quite a while now, including in the medical domain. We can leverage the power of the immense number of existing web services by integrating them into the new scenarios proposed in this paper. For example one can reuse existing web services for semantic interoperability between medical stakeholders by semantically annotating these web services.

As it will result from next chapters, the issues of semantically annotating, registering, discovering and consuming services and agents are similar. There are

thus many similarities between agents and services used for semantic interoperability. The backbone for their interoperability is the usage of ontologies.

## 2. Semantic Annotations, Publication and Discovery

Semantic annotations have been around for some time now in various forms. Probably the most familiar form of annotation is a tag, i.e. a keyword or a group of keywords attached to a piece of text published online aiming to speed up the search, helping at the same time in finding more relevant and precise information.

However, there is hardly anything *semantic* in tagging. The real semantic annotation implies going at a deeper level and enhancing the unstructured data online with a context linked with a structured domain. The ultimate goal of semantic annotation is to enable not only better aimed search, but to transform a simple information retrieval (i.e. based on textual matching) into structured data retrieval (i.e. based on concepts from a domain).

Web services are described in WSDL [4] (*Web Service Definition Language*), using a format based on XML. This description is registered into a repository (traditionally UDDI-based) and thus a web service can be discovered and consumed, as shown in **Error! Reference source not found.**. Or at least this is the theory. In practice the step of registering the service in a UDDI [5] (*Universal Description, Discovery and Integration*) repository is skipped and instead the address of the service is hard-coded into the client. Also, a proxy is generated based on the WSDL file; the proxy is then compiled together with the client of the web service and thus the client can invoke the web service using the local proxy.

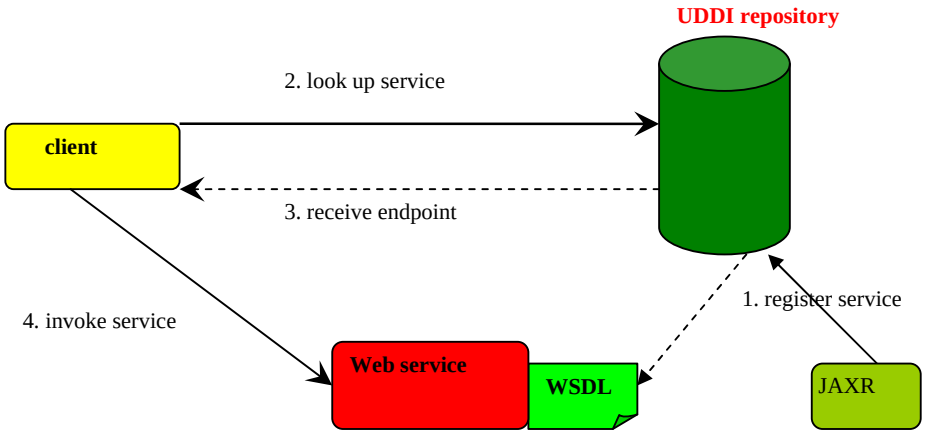


Fig. 1. Registering, discovering and consuming a web service

There are several issues with "standard" WSDL-described web services. First and foremost, WSDL is a purely syntactic representation of the web service operations with their parameters and results. WSDL doesn't say anything about the service behavior or about the semantics of its operations.

Just as one can add semantic annotations to a document, web services can also be semantically annotated. The W3C recommendation is SAWSDL [6], which stands for *Semantic Annotations for WSDL and XML Schema*, based on the older W3C submission WSDL-S. Briefly, SAWSDL provides means to semantically annotate certain elements of a WSDL.

For example, let's define an emergency service operation of giving advice based on patient's symptoms, medical record and location. This is a simplified excerpt from the WSDL description of such an operation:

```
<wsdl:binding name="EmergencyServiceSoapBinding"
  type="es:EmergencyServicePortType">

  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="GetAdvice">
   <soap:operation soapAction="http://www.emergencyservice.org/GetPatientAdvice"/>
   <wsdl:input messageLabel="In" element="Patient"/>
       <soap:body use="literal" namespace="http://schemas. emergencyservice.org
         /GetPatientAdvice.xsd"/>
   </wsdl:input>
   <wsdl:output messageLabel="Out" element="Procedure">
       <soap:body use="literal" namespace="http://schemas. emergencyservice.org
         /GetPatientAdvice.xsd"/>
   </wsdl:output>
   <wsdl:fault>
       <soap:body use="literal" namespace="http://schemas. emergencyservice.org
         /GetPatientAdvice.xsd"/>
   </wsdl:fault>
  </wsdl:operation>
 </wsdl:binding>
```

This excerpt defines the operation, with its input, output and fault messages. The actual types of the inputs, outputs and fault are defined in the XML Schema document GetPatientAdvice.xsd (but could also be embedded in the WSDL file):

```
<xsd:schema targetNamespace="http://namespaces.emergencyservice.org"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema">
      <xsd:element name="Patient">
            <xsd:complexType>
             <xsd:sequence>
              <xsd:element name="name" type="string"/>
              <xsd:element name="addr" type="string"/>
              <!— many other fields, corresponding to HL7-RIM ontology →
```

```
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        ….
      </xsd:schema>
```

This is the not annotated WSDL that describes the service. In order to give semantic meaning to this WSDL description and to the search process, one would add SAWSDL annotations, e.g.:

```
<xsd:element name="Patient">
    <xsd:complexType sawsdl:modelReference=”http://www.hl7.org/spec
      /ontology/rim#Patient”>
     <xsd:sequence>
      <xsd:element name="name" type="string"/>
      <xsd:element name="addr" type="string"/>
      <!— many other fields, corresponding to HL7-RIM ontology →
     </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

The *sawsdl:modelReference* in this example says that the Patient type of our service, maps in fact to the Patient concept from the HL7-RIM ontology (note that the URL is fictional, it's meant only to illustrates the idea).

The more semantic information we introduce in the matching process, the more accurate the matching is. There is ongoing research in the area of enhancing SAWSDL with more semantic information [7]. Typically, the extra information refers to inputs, outputs, preconditions and effects (referred to as *IOPE* from now on).

Similar to web services, an agent has a description that tells what the agent can do. The description is standardized by FIPA [8] (*Foundation for Intelligent Physical Agents*) and it is registered in a so-called Directory Facilitator (DF) [9], a kind of Yellow Pages system similar to UDDI for web services. Below we show a hospital reception agent's description. The typical service of such an agent is to receive the patient's symptoms and in turn look up a specialist within the hospital.

```
(df-agent-description
  :name
    (agent-identifier
      :name ReceptionHospitalXYZ@xyz.com
      :addresses (sequence iiop://xyz.com/acc))
  :services (set
    (service-description
      :name reception
      :type reception
      :ontology (set SNOMED)
      :properties (set
        (property
          :name “hospital reception area”
          :value 224890005)
```

```
        (property
          :name "temperature symptoms"
          :value 271399003)
      ....
       )))
  :protocol (set FIPA-Request FIPA-Query)
  :ontology (set SNOMED FIPA-Agent-Management)
  :language (set FIPA-RDF))
```

In this agent description, notice that SNOMED ontology is used. Among the interesting properties of the service offered by the hospital reception agent, we show here the "hospital reception area", which has the value 224890005 in SNOMED. Then, we enumerate all symptom types that can be sent to this agent with their corresponding SNOMED code values. These properties (i.e. "hospital reception area", the symptoms) are in fact the *annotations* we are employing in the case of agents.

A common scenario is when a personal agent looks up a hospital reception agent: all symptoms sent by the personal agent are matched and interpreted by the hospital reception agent (the "…." in the agent description above represent the symptoms according to the SNOMED terminology recognized by this agent; some of them will be filled in by the personal agent).

What we really need is *semantic matching*, as is the case with semantic web services. Our proposal is to extend the FIPA standard such that it allows that some new parameters be specified for the description of an agent registered with DF. These new parameters should be optional and should include at least the aforementioned IOPE (inputs, outputs, preconditions, effects).

Using this idea, the example hospital reception agent could be rewritten as:

```
(service-description
      :name reception
      :type reception
      :ontology (set SNOMED)
      :properties (set
        (property
          :name "hospital reception area"
          :value 224890005)
        (property
          :name "temperature symptoms"
          :value 271399003)
      ....
      )
      :inputs (set
        (input
          :name "symptoms"
          :value "http://www.ihtsdo.org/snomed/ontology#Symptom"
         )
     )
     :otputs (set
```

```
        (output
          :name "specialist"
           :value "http://www.fipa.org/agents/ontology#Specialist"
         )
       )
)
```

We've added here the *inputs* and *outputs* – this is how we propose to extend the FIPA DF standard. Also the preconditions and effects can be added in a similar way.

The previous example can be used to perform the following query: "I'm interested in the agent that represents the code 224890005 in the SNOMED ontology" (such an agent would be the hospital reception agent). With this extension (annotated IOPE), a more interesting query can be made: "I'm interested in an agent that accepts *Symptom* inputs as described by the SNOMED ontology and returns a specialist agent's address, as described by FIPA". As you can see, the latter query is more abstract than the former, since it's not constrained to a specific SNOMED code. We could even make a more abstract query, by not specifying the target ontology: "I'm interested in an agent that accepts Symptom inputs as described by my medical ontology and returns a specialist agent's contact data". In order to properly find the service described above with such a query, an ontology mapping must exists between "my medical ontology" and SNOMED. Also there must exist an ontology mapping between the specialist agent's contact data (as specified in my query) and FIPA Specialist ontology term.

However, the discovery process needs to be extended with *semantic matching of IOPE*. Semantic discovery of agents whose FIPA service descriptions have been annotated with IOPE means in fact a semantic match between the IOPE annotations of the advertised (registered) agents and the required ones. In order to do this, one of the following approaches can be taken:

- the KB (knowledge base) used by the DF needs to be extended with semantic matching capabilities for IOPE; this might include ontology matching when the ontologies of the advertised and required agent descriptions are different
- another layer of semantic matching needs to be added on top of the DF; thus, the DF would remain unchanged, but the semantic matching of IOPE would happen outside the DF
- the descriptions of the agents are not registered to DF any more, but they are instead indexed by a search engine with semantic capabilities; this is the solution we propose and it is the topic of the next section

What's important to notice is that, in order to create a semantic-based platform for interoperability between services and agents, we need to approach similar issues, solved in similar ways: both services and agents need semantic annotations of IOPEs, both need to be registered in certain repositories and be

discovered there, both need to be consumed by their clients. The backbone for interoperability is the ontologies. We propose using existing standards, such as SAWSDL with HL7 semantic annotations for web services and OWL [10] (*Web Ontology Language*) for describing the ontologies. For agents, we propose extending the FIPA agent description with annotated IOPE. Once annotated, the services and agents need to be registered and discovered in order to be consumed. In the next section we highlight our idea of registration and semantic discovery of services and agents.

## 3. Semantic Search Engine

The UDDI initiative has been the de facto standard for publishing web services. The idea is to be able to register a web service, so that it can be discovered by a client. Once it is discovered, the WSDL is retrieved and a proxy for client-service communication could be generated dynamically. This is how the UDDI has been used so far (if at all).

Moreover, mapping SAWSDL to UDDI has been researched ([11], [12]) and it's relatively straightforward:

- either directly annotate UDDI concepts with SAWSDL annotations; for example, the semantic annotations of SAWSDL are translated to UDDI *categoryBags* [11]; this is a simple approach, i.e. no semantic reasoning is needed, but some of the flexibility is lost (e.g. the search in UDDI is still syntactic)
- or semantically-enabled processing modules are layered on top of UDDI; they perform the actual semantic search – OWL ontology processing and DL reasoning

However, Microsoft, IBM and SAP dropped their support for UDDI in 2007; they were in fact the major supporters of this standard. In effect, it is time for something new, something that addresses the main limitations of UDDI:

1. Overwhelming complexity
2. Ignoring security
3. Lack of good tooling – it took us a lot of time and energy to be able to finally install locally a working UDDI implementation: jUDDI for Tomcat 3.0.1

In the authors' opinion, the future of semantic web service registration and discovery lays in *semantic search engines*, e.g. a semantic Google-like service [13]. The essential characteristics of such a semantic search engine should be:

1. simplicity – it should offer a simple, clear to use interface through which either a user or a program can search and find the desired services

2. it should offer a secure way to access a web service

3. it should offer APIs to be able to register and discover (semantic) web services

4. it should be scalable to millions of simultaneous users and yet answer within a few hundred milliseconds – like the Google search engine behaves nowadays

The idea of the semantic search engine can be extended to software agents. In the previous section we proposed annotating the agents' FIPA profiles with semantic knowledge, just the way we annotate WSDL descriptions for web services. Then we can use the same matching process for services and for agents: we perform semantic matching of inputs, outputs, preconditions and effects.

In the area of semantic matching for web services, most of the research focuses on matching based on a semantic distance between the concepts. There are many flavors of semantic matching algorithms, i.e. including only the web service *inputs* and *outputs* or taking into account *preconditions* and *effects* as well, logic-based or non-logic based matching, hybrid, etc.

In [14], the authors propose matchmaking between web services based on the so-called *Tversky model*, in which similarity is based on the properties of concepts (expressed in their given ontologies). They argue that this method offers more accurate results than the traditional methods based on semantic distance. Moreover, their method can be used with concepts from different ontologies, thus the ontology matching is also taken into account. In short, Tversky defines the semantic similarity of two concepts such that the more properties they have in common, the more similar they are. Tversky similarity measure is 1 for identical or perfectly matching concepts and 0 for completely disjoint concepts.

In this section, using the research already done, we focus on a realistic alternative to UDDI (for web services) and to FIPA DF (for agents). We envision that as a semantic search engine.

Fig. 2 shows the building blocks of the proposed semantic search engine. The three main functions of a search engine are covered by the blocks *Crawler*, *Indexer* and *Searcher*.

We've implemented a prototype of this architecture based on open source tools. The crawlers are implemented with Crawler4J [15]. We've programmed our crawlers to retrieve WSDLs from various websites, such as http://www.service-repository.com. However, note that most of the service repositories online don't contain annotated WSDL (such as SAWSDL).

Thus, we feed the crawlers manually with SAWSDL files. Note also that the SAWSDL pages typically don't have links to other SAWSDL pages (as it is the case with regular web pages).

However, the crawlers parse the SAWSDL pages looking for ontologies, pointed to by *sawsdl:modelReference* attributes. Then they also retrieve the ontology pages, e.g.:

http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder.
Every file retrieved by a crawler is brought to a repository.

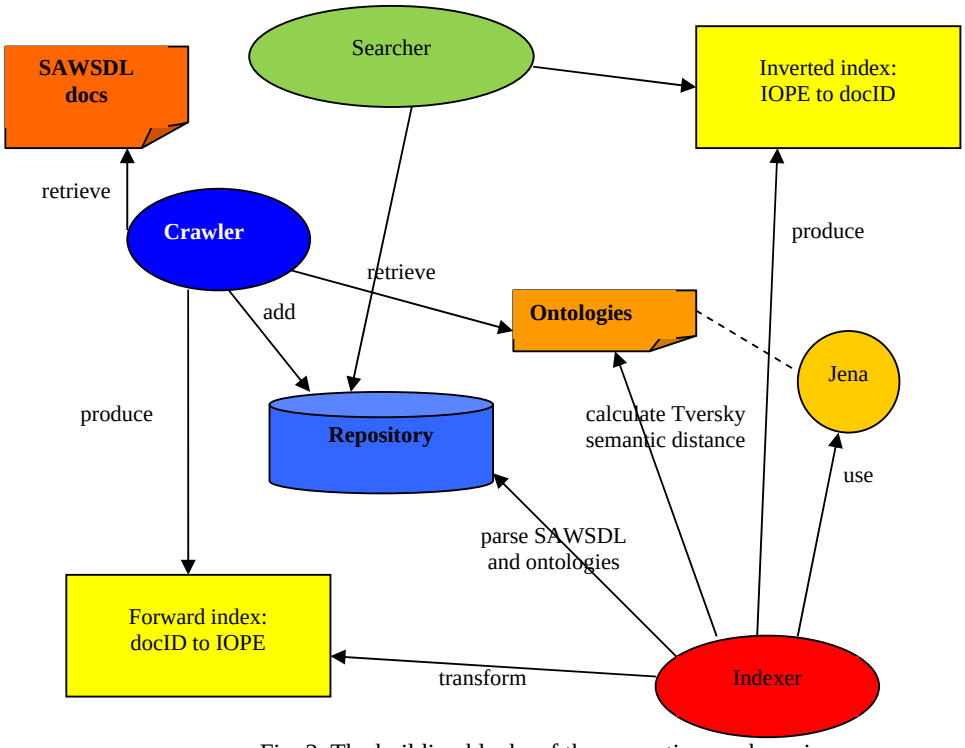


Fig. 2. The building blocks of the semantic search engine

The repository is implemented using BerkeleyDB [16], an Oracle product which can be used for free for non-commercial purposes. Its architecture is based on storing *(key, value)* pairs in efficient data structures (e.g. *B-trees*), providing high concurrency and speed along with simplicity in programming. It supports ACID transactions and offers support for *replication*, enhancing thus the availability and scalability of the data store. These characteristics make it a good candidate for implementing the repository and the indexes in a search engine system.

The crawlers use Apache Woden [17] combined with some extensions for SAWSDL to parse the WSDL service descriptions. With this we can parse WSDL 2.0 files, which is good enough for demonstration purposes. However, in the real world we need to be able to parse both WSDL 1.1 and 2.0. Luckily, one of the latest Woden versions comes with a converter from WSDL 1.1 to 2.0, which we use in our prototype. The crawlers create the forward indexes and store them in

the repository. Such an index maps the documents (WSDLs) to their inputs, outputs, preconditions and effects, respectively.

The indexers read the forward indexes and create the reverted indexes, also stored in the repository. The indexers use Jena [18], an open source inference engine, in order to pre-calculate the Tversky semantic distances for each two concepts of the retrieved ontologies and store them in the repository. This greatly enhances the performance of the searcher (whose algorithm is described in the next section).

The searcher is itself a web service written in Java, which accesses the inverted indexes (there is an index for the inputs, one for outputs, one for preconditions and one for effects) in order to find the matching WSDL files. The search algorithm is described in the next section. The key issue with the searcher is its performance, so we use the optimized data structures of BerkleyDB to create indexes as B-trees in memory and thus minimize the disk access, the most time consuming operation.

## 4. Search Algorithm

The first important requirement to the data our search engine works with is that the WSDL descriptions of the services be annotated with semantic information. Our semantic search engine deals with services whose descriptions (WSDL files) contain IOPE annotated with ontological concepts using SAWSDL. Our crawlers are fed with URLs pointing to semantically annotated WSDL documents. They fetch the documents and store them compressed in a repository. Each document is assigned a unique docID, derived from its URL. Importantly, the crawlers also fetch the ontology files used to annotate the WSDLs. Since they parse the SAWSDL files, the crawlers also create the forward indexes.

As mentioned before, for each ontology referred by the IOPE concepts, the indexer involves the inference engine Jena to calculate the Tversky matching values between each two concepts of that ontology. This list of matches is sorted in descending order by the matching values (the best matches first) and is stored together with the corresponding ontology file.

Then the forward index is converted into four inverted indexes, each corresponding to I, O, P and E respectively. For each concept from IOPE, a unique wordID is generated. The indexes are sorted by this wordID.

With these preparations in place, the search process needs to return the best matching web service for a given profile. By profile we mean a set of IOPEs representing the desired characteristics of a service. The preconditions and effects are not mandatory, but the inputs and outputs are.

The search algorithm is structured on three levels:

1. First a syntactic (textual) matching is done with the IOPEs of the given profile; this is the same as what a traditional search engine would do, but it only returns the most relevant match (preferably one that covers all IOPEs)
2. If there was no result, a first kind of semantic matching is done: for each *output* concept in the given (requested) profile, we take each *subclass* in its ontology in increasing order of the semantic distance, as defined in [19] (below we explain why we consider the subclasses for outputs and superclasses for inputs); we compute the wordID for the current subclass and look it up in the corresponding index; thus, we try to find the best matches for all outputs according to the semantic distance; if at least one output can't be matched, the algorithm moves to step 3; then we do the same for *inputs*, but now we take the *superclasses* of each given (requested) input; if we have at least one input matched, we find the matches common for the outputs and the inputs; finally, we check which of these also match the preconditions and effects of the given profile, if they exist
3. If there was no result from the previous step, more complex semantic matching is performed: for each output concept in the given profile, we take all concepts from the same ontology (except the subclasses covered at step 2) in decreasing order of Tversky distance pre-calculated for the current output; we compute the wordID for each such concept and look it up in the corresponding index; if at least one output can't be matched, the algorithm stops and no match is returned; then we do the same for inputs; if we have at least one input matched, we find the matches (services) common to the outputs and the inputs; finally, we check which of these also match the preconditions and effects of the given profile, if they exist

Note that in step 1 we do a standard, textual match. This helps when an advertised web service is annotated with a concept such as:

http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderRequest, which is also part of the required profile and where the namespace and the name of the concept match textually. This alleviates the need for a semantic match, since we can assume both the required profile and the advertised service mean the same thing.

Step 2 does a first semantic match. Note that we need all outputs of a required profile to be matched. If any output is not matched by an advertised web service, we'd remain with a partial answer, which is not acceptable, since the outputs are used to compose and further invoke web services. As for the inputs, the requirement is less strict: we need at least an input to match, not necessarily all. In order to match an input, we use the *covariance* principle: if the required

profile's input is a subclass of the advertised service's input, they match. Intuitively this means that if an advertised service can deal with a certain concept, it can also deal with its derived concepts, since they are just restrictions of the original concept. That's why in our algorithm we look for the superclasses of required inputs among the advertised inputs.

However, for the outputs we have the dual principle (*contravariance*): if an advertised service outputs a certain concept, we can never require a more restrictive version (i.e. a subclass) of that. Only a superclass of that concept will match.

Finally, if there are no matches so far, Tversky model is used, which provides a matching value for any two concepts of an ontology. We start with the best Tversky match (e.g. two concepts which differ by just a property) and then, if no match is found, we proceed with lower matches, until a certain threshold. The threshold is necessary to eliminate really bad matches and it is a parameter of the search process.

## 5. Conclusions

In this paper we've presented the building blocks of a system based on software agents and web services, which is designed to assist in improving the quality of the medical act.

The system is based on semantic capabilities, which are supposed to enhance the interoperability among medical systems.

We've tried to unify the view on agents and services by showing that both present the same challenges:

- they need to be semantically annotated; in order to do that, we need to employ ontologies
- they need to be registered in certain repositories
- they need to be semantically discovered
- they need to be consumed, i.e. they communicate

We've proposed the extensions of FIPA service description to allow annotation with IOPE (inputs, outputs, preconditions and effects), similar to the research with the same purpose for SAWSDL in the world of web services. Our prototype shows that just by adding IOPE semantic annotations, the search process is dramatically enhanced in terms of accuracy, while at the same time it is still scalable and efficient (i.e. the IOPE annotations don't affect significantly the search efficiency).

We've also proposed and prototyped a semantic search engine used typically to index WSDL service descriptions whose IOPEs are annotated according to the SAWSDL standard. The search engine can be extended to index

agent descriptions as well, thus further unifying the world of agents and the world of services.

## R E F E R E N C E S


[1] *** DICOM - http://medical.nema.org/
[2] *** HL7 - http://www.hl7.org/
[3] *** SNOMED-CT, http://www.ihtsdo.org/
[4] *** WSDL, http://www.w3.org/TR/wsdl
[5] *** UDDI version 3.0.2, http://uddi.org/pubs/uddi_v3.htm
[6] *** SAWSDL, http://www.w3.org/2002/ws/sawsdl/spec/
[7] *Y. Chabeb, S. Tata, A. Ozanne*, "YASA-M: A Semantic Web Service Matchmaker", in: 24th IEEE International Conference on Advanced Information Networking and Applications (AINA'10), pp. 966-973 (2010)
[8] *** FIPA, http://www.fipa.org/
[9] *** FIPA DF, http://www.fipa.org/specs/fipa00023/SC00023K.html#_Toc75950983
[10] *** OWL, http://www.w3.org/2004/OWL/
[11] *P. Chatel*, "Toward a semantic Web service discovery and dynamic orchestration based on the formal specification of functional domain knowledge", *International Conference on Software & Systems Engineering and their Applications (ICSSEA), 2007*
[12] *D. Kourtesis, I. Paraskakis*, "Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery", in ESWC'08 Proceedings of the 5th European semantic web conference on The semantic web: research and applications, 2008
[13] *S. Brin, L. Page*, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", Computer Networks and ISDN Systems 30, pp. 107-117, 1998
[14] *J. Cardoso, J. Miller, S. Emani*, "Web Services Discovery Utilizing Semantically Annotated WSDL", Reasoning Web, Springer-Verlag Berlin, Heidelberg, 2008
[15] *** Crawler4J, http://code.google.com/p/crawler4j/
[16] *** BerkeleyDB,, http://www.oracle.com/technetwork/database/berkeleydb/
[17] *** Apache Woden, http://ws.apache.org/woden/
[18] *** Jena, http://incubator.apache.org/jena/
[19] *P. Pukkasenung, P. Sophatsathit, C. Lursinsap*, "An Efficient Semantic Web Service Discovery Using Hybrid Matching", in Lecture Notes in Computer Science, Volume 6162, 110-119, 2010