Developing RDF-based Web Services for Supporting Runtime Matchmaking and Invocation

Hong Qing Yu, Dong Liu, Stefan Dietze and John Domingue
Knowledge Media Institute
The Open University
Milton Keynes, United Kingdom

Abstract— In our previous research, we proposed an Autonomous Matchmaking Web Services (AMWS) framework, in which Web services broker themselves to notify the service registry whether they are suitable to the service requests. This framework is designated to more efficiently work for dynamically assembling services at runtime in a massively distributed environment. The central point of the AMWS is to use RDF (Resource Description Framework) to carry all exchanging messages. However, the implementation detail has not been discussed yet. In this paper, we focus on showing the two most important implementation parts of (1) transforming existing services to become AMWS compliant services that can consume and produce RDF messages; (2) service semantics can be annotated and self-brokered by using our developed Development Time Semantic Annotation and Lowering (DtSAL) Library at programming time.

Keywords: Web services, Autonomous Matchmaking, Semantic Web, Semantic Web Services, RDF.

I. INTRODUCTION

Automatic service discovery, selection, orchestration and invocation are the main research topics in the fields of service oriented computing. The majority of research efforts towards service automation is based on Semantic Web Services and targets the issues at the semi-automatic level, in which each individual task (e.g. discovery, selection, orchestration and invocation) is completed automatically by machines but human coordinates the whole service consumption lifecycle. Figure 1 shows the common process of development lifecycle based on current ideas of Semantic Web Services. Services are annotated with semantics that are stored in the service semantic repository. When service users want to consume services, they usually first discover services through a semantic broker, then invoke the corresponding services.

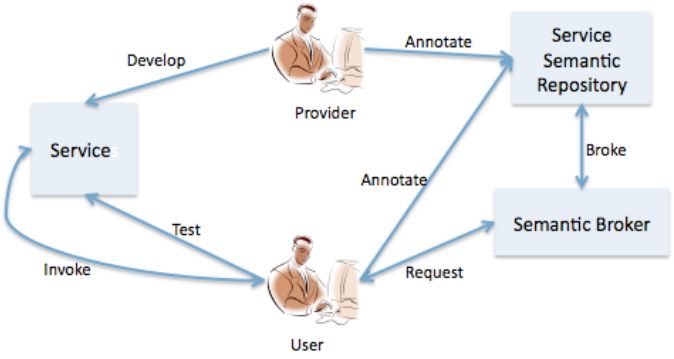

Figure 1. The Current Semantic Web Service architecture.

The work at the semi-automatic level can be further divided into two categories A and B (see Figure 2).

Category A fully trusts and depends on the Web service broker. In this way, the client is thin by just sending service specification and waiting for the expected response. The broker is a fat by taking care of all the dynamic activities. The main research contributions in this category are WSMX architecture [1] (e.g. IRS-III [2]) and OWL-S [12] based architectures (e.g. service composition application [4] based on planning algorithms). However, to be able to use these architectures, the user has to understand the data model of the services that registered in the broker and the complicated OWL or WSMO ontologies in order to invoke the discovered services and represent the output data correctly. Due to these prerequisites, all proposed orchestration managers are pre-defined static orchestration rather than runtime configurable.

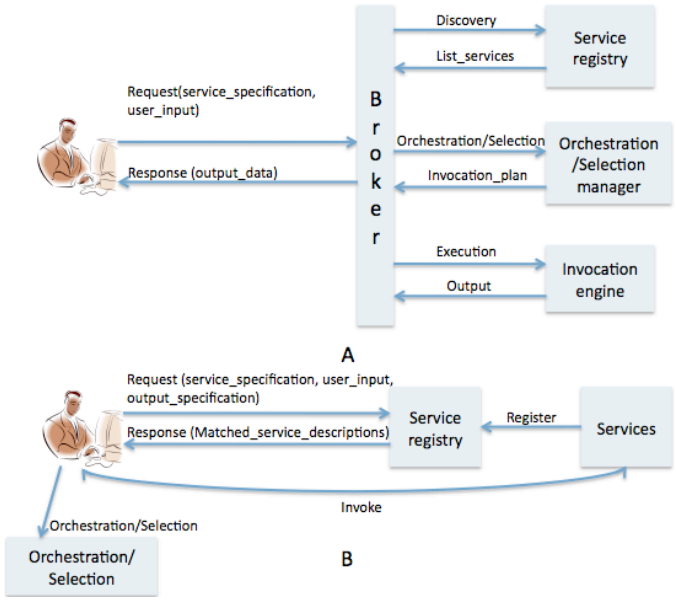

Figure 2. Two categoies of semi-automatic approaches.

Category B uses registration style to dynamically discover suitable services but leave other dynamic activities to the client side. In this way, client side can control service orchestration or select business strategy. The most recent contribution is Linked Services Concept [18] that uses Linked Data [13] principles to publish service semantics in to the Linked Data cloud.

Application [15] and [19] are developed based on Linked Services.

Semi-automation benefits to system development at design time, but it is not suitable for runtime environment in which system context changes dynamically. Furthermore, the following two issues have to be addressed in order to achieve service consumption at full-automatic level, in which the whole service consumption process lifecycle is automatically completed by machines without any human interactions.

- *Hidden data representation model* which causes the problem of sharing the common understanding of the service input and output data representations. The major reason is the contradictions between semantic description/annotation and non-semantic Web service implementation. First of all, all current service description languages focus on describing service interface rather than services themselves [26]. Secondly, no matter how services are semantically described or annotated, the underlying service invocation and response messages are still based on non-semantic invocation messages, such as SOAP [8], XML or Json[1]. Therefore, manually setting up the invocation message to map the parameters is unavoidable [22].
- *Out-of-date semantics* which leads the invocation faults. Current Semantic Web Service standards require service providers or users to register service semantics once service is implemented or used to a third party controlled service repository. However, the changes of service cannot be automatically reflected, especially for users who registered the service semantics but cannot know any changes on service side.

The main conceptual frameworks and specifications for semantically describing services (e.g. WSMO, OWL-S and SAWSDL which derive from WSDL-S [16]) are very comprehensive. Most SWS initiatives were built upon the enrichment of SOAP-based Web services to have semantics. Although some tools have been developed to foster to use these standards such as WSMO studio [20], these comprehensive semantic standards are too heavy to only show interface semantics of the service and still not describing the important part of the service – data representation model.

It is only most recently that lightweight services (e.g. Web APIs and RESTful services) and service annotations have been researched. The main results of these recent studies are SA-REST [16], hRESTs [25], WSMO-Lite [14], MicroWSMO [17] and MSM (Minimal Service Model) [21]. These standards are easily to be understood and adopted. However, current processes to use these lightweight semantics are still focusing on service annotations for implementing a big middle broker layer rather than thinking of adding semantic values directly to services themselves (e.g. iServe platform [21]). Therefore, the issues of runtime service invocation and out of date semantics are still remaining.

An Autonomous Matchmaking Web Services (AMWS) framework proposed in our previous work [9] aims to tackle the above issues and introduces a semantic message (RDF) based Web Service standard. In this paper, we move one step forward to discuss how semantic message based autonomous matchmaking Web services can be implemented. A development-time semantic annotation and lowering library is implemented to minimize the out-of-date semantics.

The remainder of this paper is organized into four sections. Section II plains the background and our motivation. Section III introduces the details of the service development. Section IV discusses the related work. Section V draws a conclusion and discusses our future research directions.

II. Overview of The Autonomous Matchmaking Web Service Framework

Our initial work [9] has introduced the fundamental concepts of the Autonomous Matchmaking Web Service (AMWS) framework (see Figure 3). Comparing to traditional Semantic Web Services approach, there are two key changes:

- using RDF-based semantic message exchanging protocols rather than syntax-based message protocols (e.g. SOAP); and
- introducing semantic query endpoint for the service.

The invocation endpoint is as same as normal service invocation endpoint but only consumes and produces RDF messages. The semantic query endpoint takes RDF service request message as inputs and responses to the user by dynamically checking whether its own semantics satisfies the request. In this way, service registry only charges to broadcast the service requests in the suitable service category and let services themselves to decide if they are the matched services.

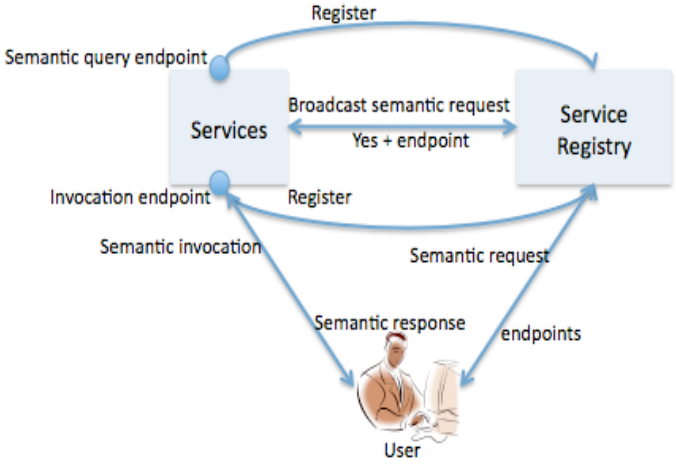

Figure 3. Autonmous Matchingmaking Web Service Framework

The AMWS framework brings at least two benefits for service-oriented computing.

- *Facilitating the full-automation of the service consumption process*: all information and communication messages are semantically understandable by using unified RDF data structure and LOD semantics. As result, the data structure and semantics are known at the same time, which is a fundamental requirement to enable services to be automatically assembled and invoked.

[1] http://json-schema.org/

- *Balancing workload among services, requester and registry*: each part of the three takes their own responsibilities to efficiently finish the service consumption life-cycle. Therefore, the AMWS framework is suitable for the large scale distributed applications.

However, current Web services standards are not support the semantic query endpoint and only have invocation endpoint. Therefore, converting existing services to enable implementing Autonomous Matchmaking Web Service Framework is the core challenge. The following section will give details of our current approach to transforming existing web services to the AMWS-compliant services.

III. Implementing AMWS-Compliant Web Services

This paper focuses on transforming existing services to become Semantic-Message-based Web services. There are following three key steps.

(1) *Transforming communication protocol to RDF*: this is done by wrapping the existing Web service functions to have only one input parameter that is a RDF message string and only one output parameter that is a RDF message string too (see Subsection A for details).

(2) *Semantic annotating services while developing*: Unlike Semantic Web Service technology, our annotation work is added while developers is wrapping the function or developing a new function. Therefore, the published semantics keeps with the latest updated functional information (see Subsection B for details).

(3) *Developing an extra function to be invoked as the semantic query endpoint*: this function implements service matchmaking algorithms and determines whether the service is matched for the request (see Subsection C for details).

A. An illustation example: Wrapping DBpedia Web service

We use the DBpedia SPARQL query RESTful service[2] as an illustration example to show the wrapping process. The detailed service information is listed in Table I.

TABLE I. The specification for DBpedia query function

Service properties	Property values
Endpoint	http://dbpedia.org/sparql
HTTP method	GET
Parameters	
format	rdf/xml, text
query	Your sparql
debug	on, off
timeout	time duration (e.g. 30seconds)

[2] http://wiki.dbpedia.org/OnlineAccess

Figure 4 shows one instance of the RDF input message based on the defined RDF schema. It contains all the corresponding parameters for lowering.

Figure 5 shows the lowering technique used to translate the DBpedia SPARQL query function call from RDF request to the actual service request. HTTP GET method is used to invoke the function. The RDFInputPaser is a key function from DtSAL (see subsection B) for lowering input RDF message based on a SPARQL statement. The fixed parameter value *format*="rdf/xml" is used to create response in RDF format. Other parameter values are defined by parsing the input RDF message. After the service call, the results from the response are translated back to RDF format.

```
<rdf:RDF>
    <dbpedia:Query rdf:ID="query">
        <dbpedia:default-graph-uri>
            http://dbpedia.org/
        </dbpedia:default-graph-uri>
        <dbpedia:query>
            Select * Where {?p a <dbpedia:Person>} Limit 10
        </dbpedia:query>
        <dbpedia:debug>off</dbpedia:debug>
        <dbpedia:timeout>30</dbpedia:timeout>
    </dbpedia:Query>
</rdf:RDF>
```

Figure 4. An example RDF input for the wrapped DBpedia service

```
public String searchDBpedia(String RDFInputMessage){
    String RDFResponse=null;
    String sparqlQuery="Select ?query ?debug ?timeout Where...";
    String endpoint="http://dbpedia.org/sparql";
    RDFInputParser rdfp=new RDFInputParser();
    HashMap<String,String> parameterIndividual=
        rdfp.parseInput(RDFInputMessage, sparqlQuery);
    String query=parameterIndividual.get("query");
    String debug=parameterIndividual.get("debug");
    String time=parameterIndividual.get("time");

    HttpClient client = new HttpClient();
    GetMethod method = new GetMethod(endpoint);

    method.setQueryString(new NameValuePair[] {
            new NameValuePair("format", "rdf/xml"),
            new NameValuePair("query", query),
            new NameValuePair("debug", debug),
            new NameValuePair("timeout", time),
    });

    try {
        client.executeMethod(method);
        RDFResponse = method.getResponseBodyAsString();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return RDFResponse;
}
```

Figure 5. Wrapping the DBpedia service to RDF-based AMWS.

B. Development-time Semantic Annotation and Lowering Library

To address the out-of-date semantic usages issue, we propose to add semantic annotations inside the code while developing or modifying service functions. To achieve this, we develop a Development-time Semantic Annotation and

Lowering (DtSAL) Java Library, whose implementation structure is shown in Figure 6. The DtSAL uses openRDF[3] and RDF2Go[4] Java library to create RDF statements. A set of service description ontologies is applied to enable developer to choose differently according to the type of wrapped service (e.g. SOAP or RESTful). In the current implementation, the service description ontologies include hRESTs, WSMO-lite, micro-WSMO and MSM. Moreover, DtSAL supports to lower the RDF document by giving a SPARQL query. The lowering function foster to both Semantic Message based Web services wrapping and new development processes.

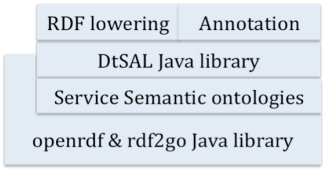

Figure 6. Implementation structure of the DTSAL Java Library

Figure 7. illustrates the annotation code for the DBpedia SPARQL query RESTful service. The annotation mainly includes four parts: (1) constructing the namespace, service name, operation name, and invocation endpoint; (2) categorising service; (3) declaring HTTP invocation method; (4) defining input parameters and their model-references; (5) defining output parameters and their model-references.

```
RestAnnotation ra = new RestAnnotation("http://www.testnamespace.com/","DBpediaSparql",
        "sparqlQuery","http://dbpedia.org/sparql");
ra.setCategory("http://amws.org/example/service/category/information/knowledge/");
ra.setHttpMethod("http://www.wsmo.org/ns/hrests#GET");

inputElement1.put("messagePart", "query");
inputElement1.put("messageMR", "http://amws.org/example/service/input/" +
        "ontology/Query#sparql");
inputElement2.put("messagePart", "debug");
inputElement2.put("messageMR", "http://amws.org/example/service/input/" +
        "ontology/Optional");
inputElement3.put("messagePart", "debug");
inputElement3.put("messageMR", "http://amws.org/example/service/input/" +
        "ontology/Testing#debug");
inputElement4.put("messagePart", "timeout");
inputElement4.put("messageMR", "http://amws.org/example/service/input/" +
        "ontology/Optional");
inputElement5.put("messagePart", "timeout");
inputElement5.put("messageMR", "http://amws.org/example/service/input/" +
        "ontology/Invocation#timeout");

outputElement.put("messagePart", "result");
outputElement.put("messageMR", "http://www.w3.org/2005/sparql-results#");

input.add(inputElement1);input.add(inputElement2);input.add(inputElement3);
input.add(inputElement4);input.add(inputElement5);ra.setParameters(input);

output.add(outputElement);ra.setOutput(output);
```

Figure 7. An hRESTs service annotation example

Figure 8 shows the DBpedia SPARQL query RESTful service annotation RDF result using hRESTs. Firstly, DBpediaSparql is defined as hrests:Service (see F.1). Secondly, sparqlQuery function is defined as an hrests:Operation of the DBpediaSparql (see F.2). Thirdly, the sparqlQuery function is categorized as knowledge that is sub-class of information (see F.3). Fourthly, hrests:Operation properties have been described including invocation method (hrests:hasMethod), invocation endpoint (hrests:hasAddress), output message (hrests:hasOutputMessage) and input message (hrests:hasInputMessage) (see F.4). Finally, all the hrests:modelReference are added to specify more accurate and specific semantic to the required domain. For instance, two model references can be used to tell that the input message includes an optional debug parameter (see F.5).

```
F. 1  <rdf:Description rdf:about="http://www.testnamespace.com/DBpediaSparql">
        <rdf:type rdf:resource="http://www.wsmo.org/ns/hrests#Service"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/DBpediaSparql/sparqlQuery">
F. 2    <rdf:type rdf:resource="http://www.wsmo.org/ns/hrests#Operation"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/DBpediaSparql">
        <hasOperation rdf:resource="http://www.testnamespace.com/DBpediaSparql/sparqlQuery"/>
      </rdf:Description>
F. 3  <rdf:Description rdf:about="http://amws.org/example/service/category/information/knowledge/">
        <rdf:subClassOf rdf:resource="http://www.wsmo.org/ns/wsmo-lite#FunctionalClassificationRoot"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/DBpediaSparql/sparqlQuery">
        <modelReference rdf:resource="http://amws.org/example/service/category/information/knowledge/"/>
        <hasAddress rdf:resource="http://dbpedia.org/sparql"/>
        <hasMethod rdf:resource="http://www.wsmo.org/ns/hrests#GET"/>
        <hasOutputMessage rdf:resource="http://www.testnamespace.com/result"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/result">
        <modelReference rdf:resource="http://www.w3.org/2005/sparql-results#"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/DBpediaSparql/sparqlQuery">
        <hasInputMessage rdf:resource="http://www.testnamespace.com/query"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/query">
F. 4    <modelReference rdf:resource="http://amws.org/example/service/input/ontology/Query#sparql"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/DBpediaSparql/sparqlQuery">
        <hasInputMessage rdf:resource="http://www.testnamespace.com/debug"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/debug">
F. 5    <modelReference rdf:resource="http://amws.org/example/service/input/ontology/Optional"/>
        <modelReference rdf:resource="http://amws.org/example/service/input/ontology/Testing#debug"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/DBpediaSparql/sparqlQuery">
        <hasInputMessage rdf:resource="http://www.testnamespace.com/timeout"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://www.testnamespace.com/timeout">
        <modelReference rdf:resource="http://amws.org/example/service/input/ontology/Optional"/>
        <modelReference rdf:resource="http://amws.org/example/service/input/ontology/Invocation#timeout"/>
      </rdf:Description>
```

Figure 8. A service RDF annotation example

C. *Adding Extra Functions to Support AMWS Framework*

By using DtSAL library, each wrapped service has its semantic to be computed in order to check whether the service can do the job according to the RDF based semantic service request message. Therefore, an extra function (semantic check endpoint) should be added to the service for answer the YES/NO to the registry by parsing the RDF request semantics and comparing to its own semantics if the service tries to support AMWS framework (see Figure 2). The response message of the semantic query endpoint follows AMWS Confirmation Response Message (CRM) standard that is represented in Figure 9. The CRM is composed by two parts of yes/no confirmation and runtime service invocation information.

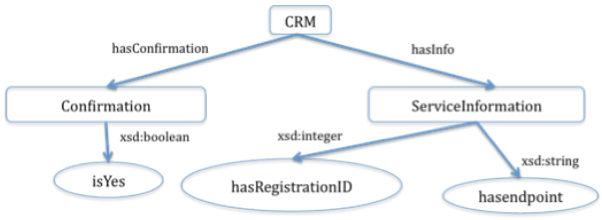

Figure 9. CRM ontology that is introduced in AMWS framework.

We implement an example of the semantic query function by using RDF2Go to parse the request semantics and checking the service semantic capability. The checking workflow is illustrated in Figure 10.

The matching algorithm used in the last step of the semantic query workflow is

[3] http://www.openrdf.org/

[4] http://semanticweb.org/wiki/RDF2Go

(1) Category is matched, if a request category term in the reference ontology is equal to or is the super-class of the service category annotation referenced term.

(2) Input is matched, if request input message terms in the reference ontologies are equal to or are the sub-class of the service input message annotation referenced terms.

(3) Output is matched, if request output message terms in the reference ontologies are equal to or super class of the service output message annotation referenced terms.

(4) Method is matched, if a request method message term in the reference ontology is equal to the service method annotation referenced term.

(5) Service should send "Yes" when all previous four conditions are matched. Otherwise, "No" will be sent.

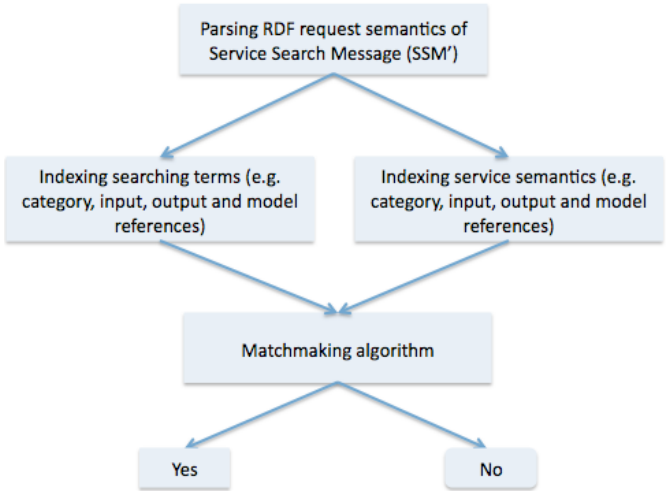

Figure 10. The semantic query workflow.

Since different kinds of services have heterogeneous business logics and concerns, the implementation of the semantic checking workflow and matchmaking workflow should be heterogeneously designed for their own purposes. Therefore, the checking workflow and algorithm illustrated here only show a possible example of the semantic checking implementation.

With both service invocation endpoint and semantic checking endpoint, the example of DBpediaSparql service is compliant to the AMWS framework to enable to be asked in broadcast way and invoked via RDF based invocation messages.

IV. Related Work

The Linked Services [21] and Linked Open Services (LOS) [22] approaches both use Linked Data theory as their technical foundations and agree that service should communicate using RDF for supporting automatic web service consumption life cycle.

The Linked Services approach focuses on annotating and publishing existing Web services semantics to the Linked Data cloud in order to discovery services using Linked Data theory. Based on Linked Service proposal, service annotation model (e.g. MSM), service annotation tools (e.g. Sweet [23] and SmartLink [24]) and semantic description repository (e.g. iServe) have been developed. Although, a preliminary service invocation engine has been developed to use the semantic data of the published services, it is still working on dynamic service invocation level because it is very handful to setting up the invocation RDF to match exactly the service invocation lowering annotations. Moreover, two more issues remains: (1) the underlying services are not added semantic values to themselves as the semantics are stored in third party repository that need to be manually updated when changing takes place; (2) centralised service semantic repository leads to service discovery and invocation are also through the centralised environment, therefore, scalability issue cannot be resolved easily.

The LOS approach focuses on wrapping existing RESTful services or SPARQL endpoints to manipulate services that can consume and produce RDF messages by using SPARQL language such as Ask or Construct to support dynamic lowering and composition. Therefore, LOS service semantic model mainly described SPARQL graph pattern of input and output. As far as our best knowledge, LOS does not support to publish service semantics and dynamic service discovery methodology. Moreover, LOS again only supports dynamic service invocation rather than automatic because, the service users need to know the lowering schema first in order to enable LOS understand their RDF invocation requests.

V. Conclusion and Furture Work

In our previous research work, AMWS framework is proposed to fully use Semantic Web technology to exchanging messages between Web service communication protocols. The advantages of AMWS are fully supporting automatic service discovery and invocation at runtime. However, the feasibility and implementation details have not been investigated. In this paper, we introduce an implementation process to wrap existing Web services (DBpedia SPARQL query RESTful service in our case) to become an AMWS framework compliant semantic message based Web service. The wrapping process includes (1) reengineering Web service to enable receiving and sending RDF semantic input and output messages; (2) annotating services on Development Time using DtSAL library and (3) developing and adding the service semantic checking function to the service.

To fully support the whole working process to the AMWS framework, many researches and implementations are remained. We list three priorities in below:

- investigating the optimization algorithm to deal with the service selection issue caused by broadcasting discovery methodology.
- developing an automatic mediation engine to mediate different service annotation referenced terms from different annotation ontologies. One possible way is to enrich the Service Searching Message (SSM) with the medication results.

- developing different DtSAL libraries to different kinds of current Web service programming language (e.g. C#).

ACKNOWLEDGMENT

The authors thank to EU funded Multi-type Content Repurposing and Sharing in Medical Education (mEucator) ECP2008EDU418006 project for supporting this work.

REFERENCES

[1] D. Roman, J. de Bruijn, A. Mocan, H. Lausen, J. Domingue, C. Bussler, and D. Fensel, "WWW: WSMO, WSML, and WSMX in a nutshell", *Proc of The Semantic Web (ASWC 2006).* LNCS, 2006, pp. 516–522.

[2] J. Domingue, L. Cabral, S. Galizia, V. Tanasescu, A. Gugliotta, B. Norton, and C. Pedrinaci, "IRS-III: A broker-based approach to semantic Web services", *Web Semantics*. 2008. vol. 6 no. 2 pp. 109-132.

[3] D. Martin, M. Paolucci, S. Mcilraith, M. Burstein, D. Mcdermott, D. Mcguinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara "Bringing semantics to web services: The OWL-s approach", *Proc of 1st Internation workshop of Semantic Web Services and Web Process Computing (SWSWPC 2004).* LNCS, 2004, vol. 3387 pp. 26–42.

[4] M. Klusch and A. Gerber, "Fast Composition Planning of OWL-S Services and Application", *Proc of the European Conference on Web* Services *(ECOWS '06).* IEEE Computer Society, 2006, pp. 181-190.

[5] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: A Research Roadmap", *International Journal of Cooperative Information Systems*, 2008, vol. 17, no. 02. 223.

[6] H. Gunzer and S. Engineer, "Introduction to Web Services", *Borland Developer Network*, 2002.

[7] C. Pautasso, O. Zimmermann, and F. Leymann. 2008. "Restful web services vs. 'big' web services: making the right architectural decision", *Proc of the 17th international conference on World Wide Web*. ACM, 2008, pp. 805-814.

[8] H. Wang; Y. Tong; H. Liu; T. Liu; , "Application-aware Interface for SOAP Communication in Web Services," *Proc of IEEE International Conference on Cluster Computing, 2006*, pp.1-8.

[9] H.Q. Yu, S. Dietze, N. Benn, "Autonomous Matchmaking Web Services," *Proc of International Conference on Computer Information Systems and Industrial Management Applications (CISIM), 2010*, pp. 420-425.

[10] J. Kopecký, T. Vitvar, C. Bournez. and J. Farrell, "SAWSDL: Semantic Annotations for WSDL and XML Schema," *IEEE Internet Computing*, 2007, vol. 11, no. 6, pp. 60-67.

[11] WSMO Working Group (2004), D2v1.0: Web service Modeling Ontology (WSMO). WSMO Working Draft, (2004). (http://www.wsmo.org/2004/d2/v1.0/).

[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic Markup for Web Services", *W3C Member submission*. 22 November 2004.

[13] C. Bizer, T. Heath, T. Berners-Lee, "Linked data - The Story So Far", *Special Issue on Linked data, International Journal on Semantic Web and Information Systems (IJSWIS),* 2009, vol 5 no 3 pp. 1-22.

[14] WSMO-lite, DOI= http://cms-wg.sti2.org/TR/d11/v0.2/.

[15] H.Q. Yu, N. Benn, S. Dietze, R. Siebes, C. Pedrinaci, D. Liu, D. Lambert, and J. Domingue, "Two-staged approach for semantically annotating and brokering TV-related services", *Proc of The IEEE International Conference on Web Services (ICWS 2010)*, IEEE, 2010, pp. 497-503.

[16] A. P. Sheth, K. Gomadam, and A. Ranabahu, "Semantics enhanced services: Meteor-s, SAWSDL and SA-REST", *IEEE Data Eng*. 2008, Bul l., 31(3):8–12.

[17] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Supporting the creation of semantic restful service descriptions". *Proc of Workshop: Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2) at 8th International Semantic Web Conference*. 2009.

[18] C. Pedrinaci, J. Domingue, and R. Krummenacher, "Services and the Web of Data: An Unexploited Symbiosis", *Proc Workshop: Linked AI: AAAI Spring Symposium "Linked data Meets Artificial Intelligence"*. 2010.

[19] D. Lambert, and H.Q. Yu, "Linked Data Based Video Annotation and Browsing for Distance Learning", *Proc of Workshop: SEMHE '10: The 2nd International Workshop On Semantic Web Applications In Higher Education*, Southampton, UK, 2010.

[20] N. Loutas, L. Giantsiou, A. Simov, V. Peristeras, K. Tarabanis, "A Tool for Annotating Services with Domain Specific Semantics", *Proc of 2008 IEEE International Conference on Semantic Computing*. 2008, pp.448-455.

[21] C. Pedrinaci, D. Lambert, M. Maleshkova, D. Liu, J. Domingue, and R. Krummenacher, "Adaptive Service Binding with Lightweight Semantic Web Services Adaptive Service Binding with Lightweight Semantic Web Services", *in eds. Schahram Dustdar and Fei Li, Service Engineering: European Research Results*. Springer, 2010.

[22] R. Krummenacher, B. Norton, A. Marte, "Towards linked open services and processes", *Proc of the Third future internet conference on Future internet*. 2010, pp. 68-77.

[23] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Semantically Annotating RESTful Services with SWEET", *Demo at 8th International Semantic Web Conference*, Washington D.C., USA, 2009.

[24] S. Dietze, S., H.Q. Yu, C. Pedrinaci, D. Liu, and J. Domingue, "SmartLink: a Web-based editor and search environment for Linked Services", *Demo at 8th Extended Semantic Web Conference*, Heraklion, Greece, 2011.

[25] J. Kopecky, K. Gomadam, T. Vitvar, "hRESTS: An HTML Microformat for Describing RESTful Web Services," *Proc of IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '08).* , 2008, pp.619-625.

[26] E. Wilde, "What are you talking about?," *Proc of IEEE International Conference on Services Computing (SCC 2007*). 2007, pp. 256-261.