Article

A Semantically Automated Protocol Adapter for Mapping SOAP Web Services to RESTful HTTP Format to Enable the Web Infrastructure, Enhance Web Service Interoperability and Ease Web Service Migration

Sean Kennedy [1,*]**, Owen Molloy** [2]**, Robert Stewart** [1]**, Paul Jacob** [1]**, Maria Maleshkova** [3] **and Frank Doheny** [1]

[1]   Athlone Institute of Technology, Athlone, Ireland; E-Mails: rstewart@ait.ie (R.S.);
    pjacob@ait.ie (P.J.); fdoheny@ait.ie (F.D.)

[2]   National University of Ireland, Galway, Ireland; E-Mail: owen.molloy@nuigalway.ie

[3]   Knowledge Media Institute, Milton Keynes, MK7 6AA, UK; E-Mail: m.maleshkova@open.ac.uk

*   Author to whom correspondence should be addressed; E-Mail: skennedy@ait.ie;
    Tel.: +353-0-87-2670380; Fax: +353-0-90-646-8148.

Abstract: Semantic Web Services (SWS) are Web Service (WS) descriptions augmented with semantic information. SWS enable intelligent reasoning and automation in areas such as service discovery, composition, mediation, ranking and invocation. This paper applies SWS to a previous protocol adapter which, operating within clearly defined constraints, maps SOAP Web Services to RESTful HTTP format. However, in the previous adapter, the configuration element is manual and the latency implications are locally based. This paper applies SWS technologies to automate the configuration element and the latency tests are conducted in a more realistic Internet based setting.

Keywords: Semantic Web Services; REST; SOAP; Web Services

1. Introduction

The Web is designed for humans *i.e.*, web pages are designed to be read and understood by humans. The Semantic Web, on the other hand, is designed for computers *i.e.*, the information content is structured so that machines can understand it. The Semantic Web is a set of technologies, which

provide "semantics" or meaning to Web content. On the Semantic Web, information is represented as a set of assertions called *statements*. Statements consist of three parts (or *triples*): subject, predicate and object (in that order). The subject is the concept the statement describes and the predicate is the relationship between the subject and the object. RDF, a W3C specification, is the Semantic Web's data model for representing statements. RDF can be modeled abstractly as a graph. RDF identifiers are URI references and RDF's official standard syntax is RDF/XML [1].

RDF's strength is describing information. Flexible though it is, RDF lacks explicit support for specifying the semantics behind the descriptions [2]. RDFS (RDF Schema) and the Web Ontology Language (OWL) provide primitives with defined meaning thereby enabling semantics to be added to RDF statements [3,4]. RDFS provides a type system for RDF. RDFS provides mechanisms to specify classes, properties and hierarchies of both. OWL builds on top of RDFS and provides extra primitives to add further semantics. RDFS and OWL enable ontologies to be constructed. "*An ontology defines a set of representational primitives with which to model a domain of knowledge*" [5]. The primitives are typically classes, attributes and the relationships between class members.

Summary of previous contributions [6]:

This paper extends previous work in which the following conclusions are detailed:

- Implemented, tested and demonstrated a client-side configuration wizard and protocol adapter, collectively named StoRHm (SOAP to RESTful HTTP mapping). Note that the protocol adapter is unchanged between StoRHm v1 (outlined in [6]) and StoRHm v2 (outlined in this paper); where necessary, the version numbers will be used to differentiate between them.
- StoRHm transforms opaque SOAP messages to visible RESTful format supporting all of REST's constraints, enabling the Web and its inherent efficiencies.
- StoRHm enables SOAP clients to interoperate with RESTful WS.
- StoRHm is a gradual migration enabler from SOAP to RESTful WS.
- The protocol adapter in StoRHm imposes a 6–7% time penalty and relates to deployment of the adapter on a local machine.

There are two constraints imposed:

- RESTful Web Services responding to PUT/POST receive the SOAP payload untouched.
- Complex SOAP requests which map to more than one logical URI will be mapped to POST with the SOAP payload passed on untouched.

The problem and motivation for the solution

The manual nature of the configuration wizard implemented in [6] is outlined in Figure 1:

Figure 1. Manual configuration wizard [6].
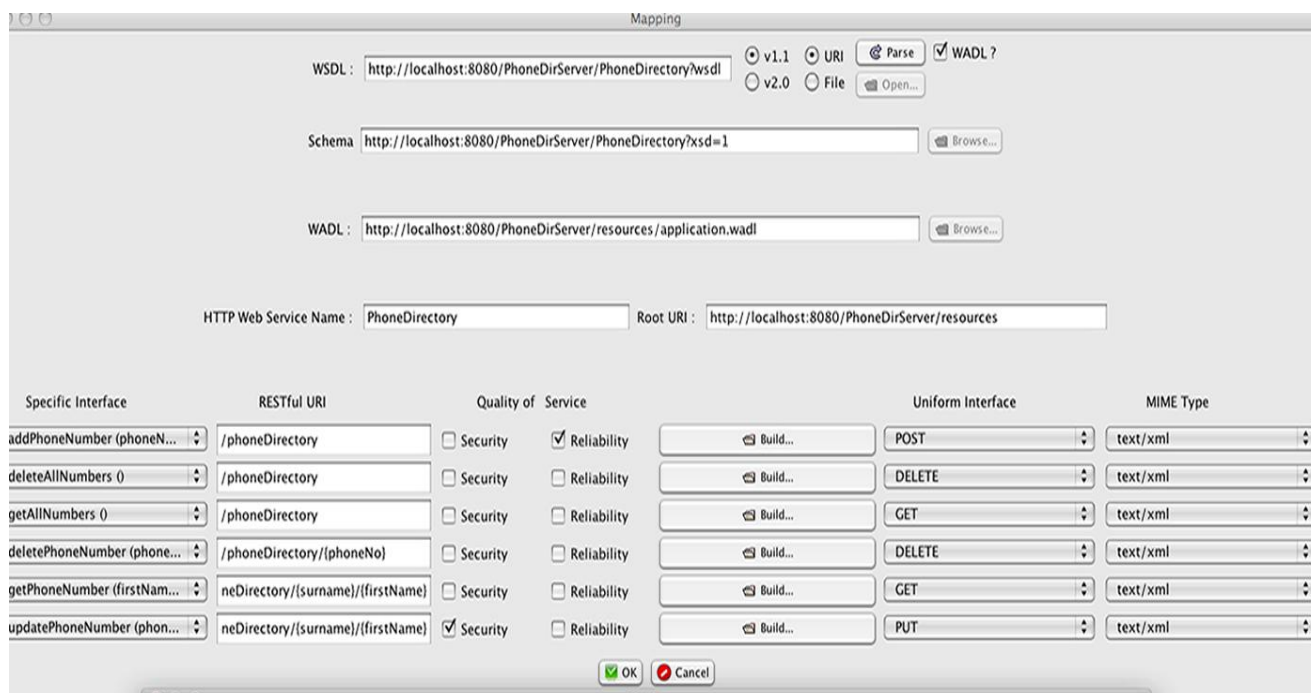
The issue with Figure 1 is that the user must manually type in a lot of data, select multiple drop down list boxes and select various check boxes and radio buttons. This is both time consuming and error prone.

Our hypothesis is that a state of the art solution leveraging SWS technologies would automate this element. This would reduce the time spent and errors made in generating the mapping file. In addition, as the protocol adapter element of StoRHm was tested on a local machine, the authors wish to determine its performance when deployed on the Internet.

The remainder of this paper is organised as follows: Section 2 is Related Work, Section 3 is the Technology Overview, Section 4 describes StoRHm v2, Section 5 is the Testing section, Section 6 is the Evaluation section and lastly, Section 7 outlines our Conclusions and Future Work.

2. Related Work

In this section, we discuss integrating both SOAP and RESTful WS into the Semantic Web. An overview of alternative RESTful WS semantic annotation is also given. Lastly, intrinsic SWS features related to our work *i.e.*, data mediation and automation are also discussed.

2.1. Integration of Web Services into the Semantic Web

In [7], the authors outline Semantic Bridge for Web Services (SBWS). SBWS is a framework that wraps around both a SOAP WS (WSDL) description and a RESTful WS (WADL, see below) description creating a SPARQL endpoint for those services. The SOAP and RESTful WS descriptions are semantically annotated, thereby enabling SBWS to execute SPARQL queries against them. SBWS analyses the SPARQL queries to determine what Web Service or combination of Web Services, will provide the answer.

From a traditional WS perspective *i.e.*, SOAP WS, both StoRHm and SBWS describe the SOAP WS using WSDL. However, to semantically annotate the WSDL file, SBWS uses OWL-S, a W3C submission since 2004 [8]. Our solution on the other hand, uses SAWSDL, a W3C recommendation since 2007 [9]. The contrast between the SBWS and StoRHm stacks is demonstrated in Figure 2.

Figure 2. SBWS stack compared to the StoRHm stack for annotation of SOAP WS.
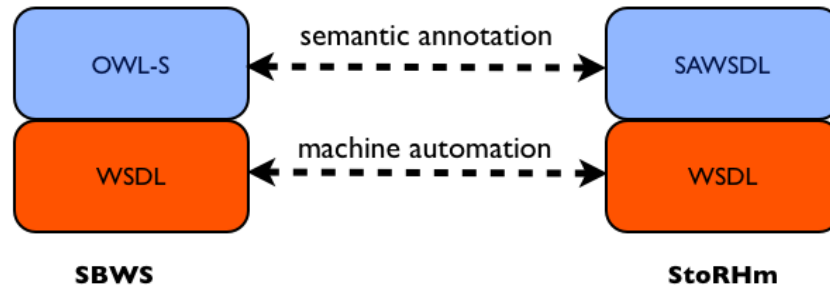
As regards the RESTful WS description, SBWS uses a WADL (Web Application Description Language) file. WADL is a simple alternative to WSDL for use with XML/HTTP applications. However, it is argued that the most popular format for describing RESTful WS is plain (X)HTML [10,11]. Consequently, our solution use plain (X)HTML to describe the RESTful WS. In order to make the unstructured HTML machine-processable (as per WSDL), we annotate the (X)HTML with a microformat called hRESTS (HTML for RESTful Services) [11]. A microformat is a way "*to add semantic metadata to human readable text in such a way that machines can glean the semantics*" [10]. The hRESTS service model is very similar to the WSDL service model; so much so, that the authors state that "*hRESTS is roughly equivalent to WSDL*" [11].

To semantically markup the RESTful WS description, SBWS inserts *custom* annotations in the WADL file. In contrast, we use MicroWSMO, a lightweight extension to hRESTS to semantically annotate our (X)HTML RESTful WS description [11]. MicroWSMO, a microformat based on SAWSDL, is to hRESTS, what SAWSDL is to WSDL. The contrast between the SBWS and StoRHm stacks is demonstrated in Figure 3.

Figure 3. SBWS stack compared to the StoRHm stack for annotation of RESTful WS.
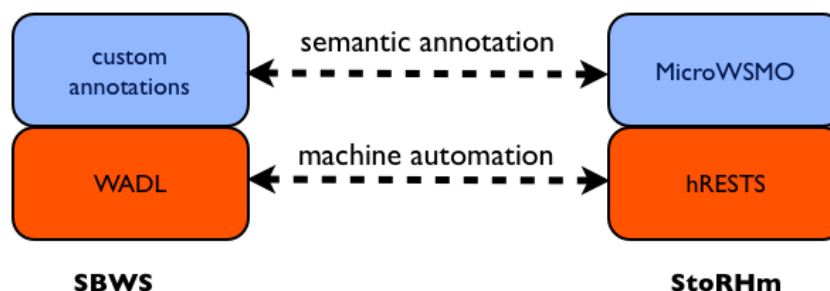
2.2. RESTful WS Semantic Annotation Alternatives

As stated above, from the RESTful WS perspective, StoRHm v2 uses both hRESTS and MicroWSMO to achieve machine automation and semantic annotation respectively. These technologies are discussed in more detail in Section 3. Other available alternatives are discussed now:

GRDDL: GRDDL (Gleaning Resource Descriptions from Dialects of Languages) is a W3C standard that enables an author to markup an XHTML file with any microformat and specify the transformation (an XSL file) that extracts the RDF based on that microformat [12]. GRDDL uses the XHTML *profile* attribute to specify that this XHTML page contains GRDDL annotations. GRDDL aware parsers will then look for the *rel* and *src* attributes on the *link* element to locate the transformation document to apply.

RDFa: RDFa (RDF in Attributes) is also a W3C standard [13]. Using both existing and new XHTML attributes, RDFa enables the insertion of RDF statements directly into an XHTML page. The values of these attributes can refer to concepts from *any* independently defined vocabulary thereby giving RDFa great flexibility. Parsers can generate RDF files from RDFa-annotated XHTML. SA-REST (see Section 2.3) uses RDFa to annotate service descriptions with its vocabulary of terms [10].

GRDDL and RDFa both have their advantages and disadvantages. The advantage of GRDDL is that it is less intrusive than RDFa *i.e.*, GRDDL only requires two lines of code in the *head* section of the XHTML document. However, two files must now be maintained: the annotated XHTML file and the transformation file. RDFa has the advantage of being a standardized microformat. In addition, RDFa is free to use any vocabulary of terms and requires maintenance of only one file: the annotated XHTML file.

Essentially the decision is between microformats (hRESTS/MicroWSMO) and RDFa (SA-REST/RDFa). However, at the time StoRHm v2 was being developed, the dearth of tools supporting RDFa/SA-REST was instrumental. By contrast, a lightweight, browser-based suite of tools, under the umbrella term Core Dashboard was available for hRESTS/MicroWSMO annotation [14].

2.3. Automation and Data Mediation

An ontology is "*a conceptualization of a domain represented in terms of concepts and the relationships between those concepts*" [10]. Moreover, ontologies provide a common nomenclature to improve interoperability. The ontology layer is the higher conceptual model where disparate Web Services are matched *i.e.*, two concepts match if they have been annotated with the same ontology concept [10]. Matching is how automation is enabled.

Integration of discrete Web 2.0 service datasets is known as a "mashup". The creation of a mashup is a non-trivial exercise that requires the use of tools e.g., Yahoo! Pipes [15]. The tools are however, customized for specific services and/or data formats and to ensure seamless integration *i.e.*, no manual intervention, a semantic mashup (or "smashup") is required [10]. The authors in [10] propose SA-REST (Semantic Annotation of Web Resources, formerly Semantic Annotation of RESTful WS). By semantically annotating the RESTful WS description, automation of matching and data mediation is enabled.

Data mediation using semantic technologies is best explained with the aid of an example. Figure 4 outlines how the semantic layer ontology is used to facilitate data mediation. Let us assume that the services in Figure 4 are address-based e.g., the SOAP WS outputs its address in one line of XML and that the RESTful WS requires its input address over two lines in JSON format. The concepts, once they are matched at the semantic layer can mediate the data. This is done as follows: the SOAP service provides a "lifting" schema (XSL transformation) to transform the XML message into an ontology-compliant data structure. The RESTful WS provides a "lowering" schema to transform the common ontology data structure into the input format it requires.

Figure 4. SOAP WS output mapped to RESTful WS input.
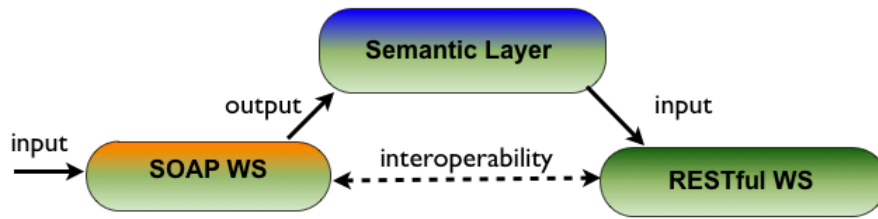
At this point it is worth pointing out a novel use in our solution of the SWS technology stacks outlined in Figure 4. The novelty in StoRHm v2 is the *use* of the stacks. These stacks were invented for mediation and integration within and across the SOAP and RESTful HTTP paradigms *i.e.*, to "*...chain together the output from one Web service to the input of another Web service*" [7]. As explained above, Figure 4 demonstrates the mapping of a SOAP WS output up to the (common) semantic layer so that the output can then be mapped to the input format required by the RESTful WS. Rather than mediate an output response from a SOAP WS to the input of a RESTful WS, we are mediating the SOAP WS *input* to the RESTful WS input. This is demonstrated in Figure 5. The complete replacement of the SOAP message with its RESTful HTTP counterpart was not envisaged when the stacks were created.

Figure 5. SOAP WS *input* mapped to RESTful WS input.
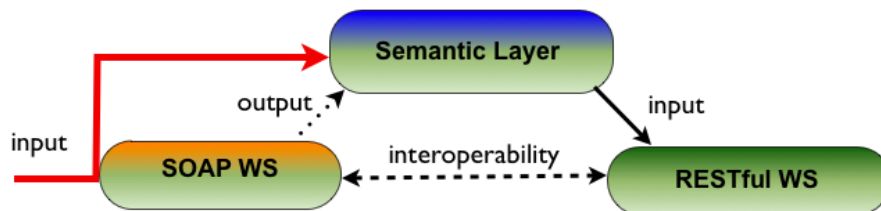

3. Technology Overview

In this section, we give an overview of Semantic Web Services.

3.1. Semantic Web Services (SWS)

Semantic Web Services (SWS) are Web services enriched with semantic metadata in order to facilitate dynamic WS discovery, selection, composition, invocation and automation [16]. MicroWSMO, SAWSDL (Semantic Annotations for WSDL and XML Schema) and WSMO-Lite are lightweight approaches to semantic annotation. Figure 6 shows the integration of the SWS technologies into both the SOAP WS and RESTful WS stacks and their cross-layer inter-relationships. The SWS stacks in Figure 6 support composability, integration and mediation within and across the stacks.

Figure 6. Semantic Layers [11]. SOAP-based WS are on the left and RESTful WS are on the right.
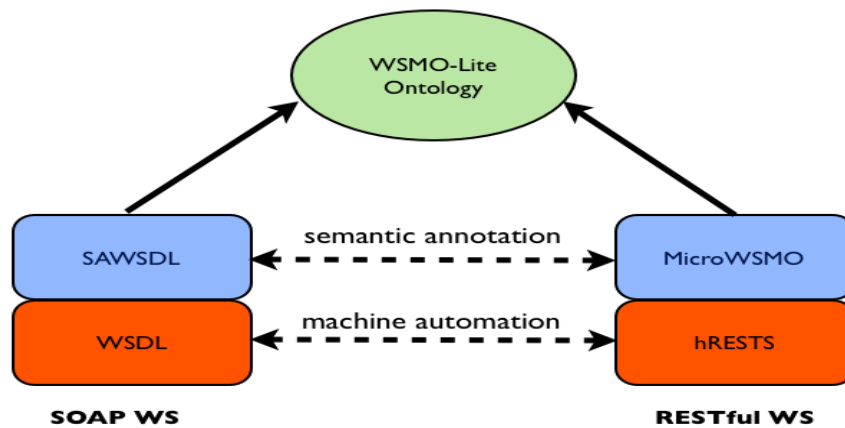
SAWSDL

SAWSDL is a specification for semantically annotating relevant elements in the WSDL file [9]. With WSDL, one is concerned with service signature, service location and the protocol to use when invoking the Web service. SAWSDL, on the other hand, is concerned with mapping elements from WSDL to a higher conceptual level *i.e.*, the ontology level. To achieve this SAWSDL defines the following extension attributes [9]:

- *modelReference*–associates, via a URI, a WSDL component with a semantic concept. This attribute is typically used on element declarations, type definitions, interfaces (or port types) and operations.
- *liftingSchemaMapping*–added to element declarations and type definitions for specifying the mapping from XML to the semantic layer.
- *loweringSchemaMapping*–added to element declarations and type definitions for specifying the mapping from the semantic layer to XML.

The *liftingSchemaMapping* and *loweringSchemaMapping* attributes support data mediation between Web services. Take for example, two Web services that are trying to communicate where the output from the first does not match the input of the second. The output from the first Web service can be lifted to the ontology layer by its *liftingSchemaMapping*. The second Web service, which is pointing at the *same* ontology concept, provides a *loweringSchemaMapping* that lowers from the ontology to the input format it requires [11].

hRESTS

RESTful Web service descriptions are generally described in plain, unstructured HTML [10,11]. hRESTS (HTML for RESTful Services) is a microformat aimed at making these HTML based Web APIs machine readable by annotating the key information on the HTML page. hRESTS uses the existing class and rel attributes of XHTML to identify the key information of the service description "*effectively creating an analogue of WSDL*" [11]. hRESTS defines (XHTML) classes such as:

- *service*
- *label*
- *operation*
- *method* (the HTTP method used e.g., GET)
- *address* (the URI used; may be a URI template)
- *input*
- *output*

MicroWSMO

MicroWSMO is an extension of hRESTS that adds semantic annotations [11]. As explained earlier, SAWSDL is an extension to WSDL that specifies how to annotate service descriptions with semantic information. Given that the hRESTS service model is so similar to the WSDL model, it is not surprising that MicroWSMO is very similar to SAWSDL. MicroWSMO defines three link relations *i.e.*, anchors with the rel attribute set as follows:

- *model*–the *href* attribute points to an ontology concept or instance.
- *lifting* and *lowering*–the *href* attribute points to the transformations to and from the semantic layer respectively.

WSMO-Lite

WSMO-Lite is used to specify the actual semantic layer ontology [17]. Languages such as RDF, RDFS and OWL are used at this layer. WSMO-Lite captures four aspects of service semantics [17]:

- *Information model*–defines the data model of the service.
- *Functional semantics*–what does the service do when it is invoked. This can be specified by either:
  - o *Categorisation*–simple functionality taxonomies where functionality is organised into a hierarchy of categories. WSMO-Lite offers the RDFS class *wsl:FunctionalClassificationRoot* to distinguish functional classification hierarchies from normal ontology hierarchies.
  - o *Capability*–preconditions and effects. In [17], a telecoms example has a precondition that the client has to have a minimal bandwidth before the service can be executed and the effect identifies the valid outputs as a result of successfully executing the service. WSMO-Lite offers the RDFS class *wsl:Capability* and the properties *wsl:hasPrecondition* and *wsl:hasEffect* so that preconditions and effects can be setup in an ontology.
- *Non-functional semantics*–these are incidental details specific to the implementation (or running environment) of a service, independent of the central purpose of the service but necessary for successful completion. Examples would be the price of the service or QoS guarantees. Non-functional semantics are often used for ranking e.g., which service is the cheapest. Non-semantic languages such as WS-Policy are often used to express non-functional characteristics. WSMO-Lite provides the class *wsl:NonfunctionalParameter* to mark items with non-functional semantics.
- *Behavioural semantics*–defines the sequence of operations that a client needs to follow when invoking a service. This is done in WSMO-Lite by using functional semantics (categories and/or preconditions and effects). The URI's of these pieces of functional descriptions are attached to

the operation (in either SAWSDL as *modelReference* or MicroWSMO as *model*). The client can now reason about which operation can be executed at a particular point in time (*i.e.*, at a particular state in the application).

4. StoRHm v2

In this section, we outline StoRHm v2: its requirements, architecture, design and implementation. In addition, the research methodology used is discussed.

4.1. Requirements

The goal of StoRHm v2 is to avail of Semantic Web technologies in order to automate the configuration element of StoRHm v1. In addition, the requirements of StoRHm v1, outlined in [6] must be maintained.

4.2. Architecture

Figure 7 outlines the architecture of StoRHm v2. StoRHm v1 elements are highlighted with a blue background. StoRHm v2 elements are highlighted in red. The configuration wizard is a front-end to the mapping file and takes as input:

- a SAWSDL file representing the semantically annotated SOAP WS description.
- a MicroWSMO file representing the semantically annotated RESTful WS description.
- a shared/common ontology outlining the QoS supported by the WS.

Figure 7. StoRHm v2 Architecture.
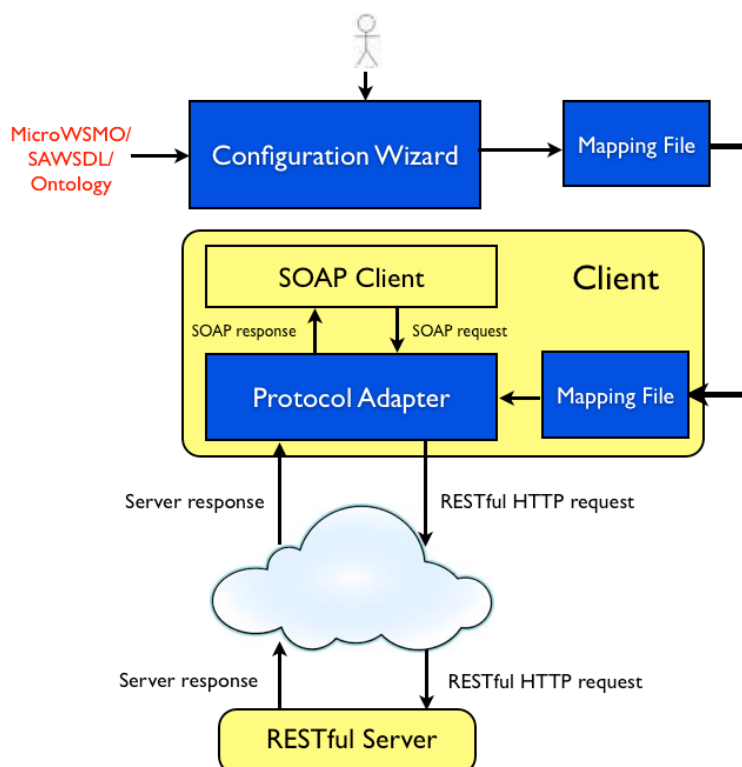
4.3. Design Decisions

As per StoRHm v1, the mapping file is implemented as a CSV file (Comma Separated File). This CSV file is subsequently used by the runtime protocol adapter to map SOAP WS request to RESTful HTTP format [6].

4.4. Implementation

The Semantic Web is a promising approach for Web Service selection and automation because RDF triples can link concepts defined in different vocabularies thereby *establishing relationships between vocabularies* [18,19]. As a result, the SAWSDL file, representing the SOAP Web Service; the MicroWSMO file, representing the RESTful Web service and the ontology, stating the QoS supported by the Web Service are all semantically annotated a priori.

The tool used to annotate the WSDL description with SAWSDL annotations is Core Dashboard [14]. Core Dashboard is a suite of tools developed by the Knowledge Media Institute at Milton Keynes in England. These tools enable semantic annotation of Web Service descriptions, both the WS-* and RESTful variety. One of the tools hosted on Core Dashboard is SWEET (Semantic Web sErvice Editing Tool) [20]. SWEET was used for annotating the RESTful WS description (an HTML file) with both hRESTS and MicroWSMO markup. The ontology editor, Protégé, was used to create the ontology [21].

A desktop-based wizard (Figure 8) supports the configuration function. The wizard is a front-end GUI for the CSV file creation. The wizard prompts the user for the names of the SAWSDL, MicroWSMO and ontology files. The files can be URI based or file based.

Figure 8. StoRHm v2 Configuration Wizard.
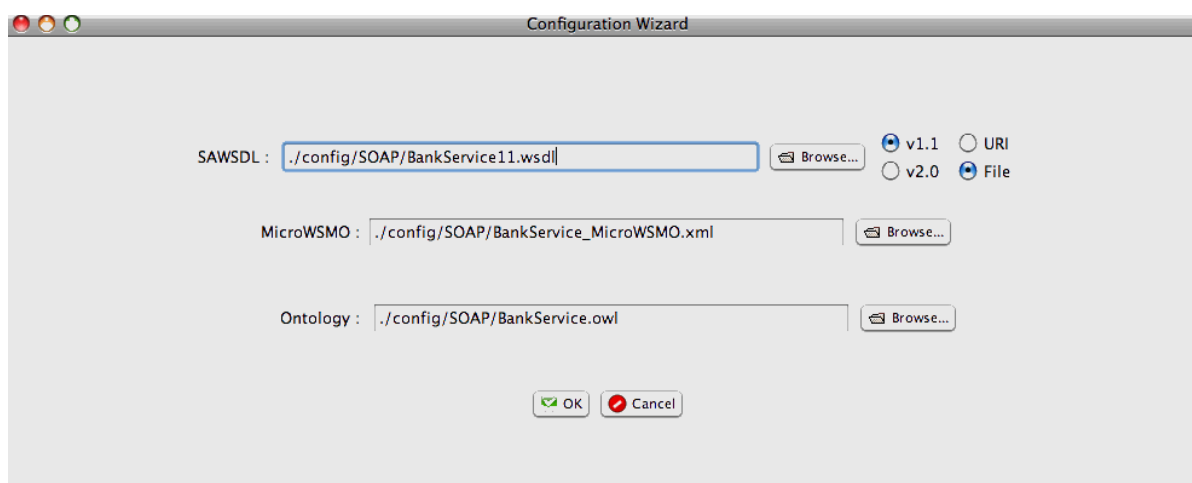
The user selects OK (see Figure 8) and the wizard obtains the model references (representing the semantic concepts) of the operations supported and their arguments by parsing the SAWSDL file. These semantic concepts are located in the MicroWSMO file in order to ascertain the URI and HTTP verb used by the equivalent RESTful Web Service. The operation concept is located in the ontology to determine whether security and/or reliability are required. The CSV file (identical to StoRHm v1 [6]) is then populated automatically. However, if any of the SAWSDL concepts are not located in either the

MicroWSMO and/or ontology files, an error is reported to the user and the configuration process exits without creating the CSV file. Figure 9 is a flow diagram detailing the implementation logic executed when the user selects OK.

Figure 9. StoRHm v2 Configuration Wizard flow diagram.
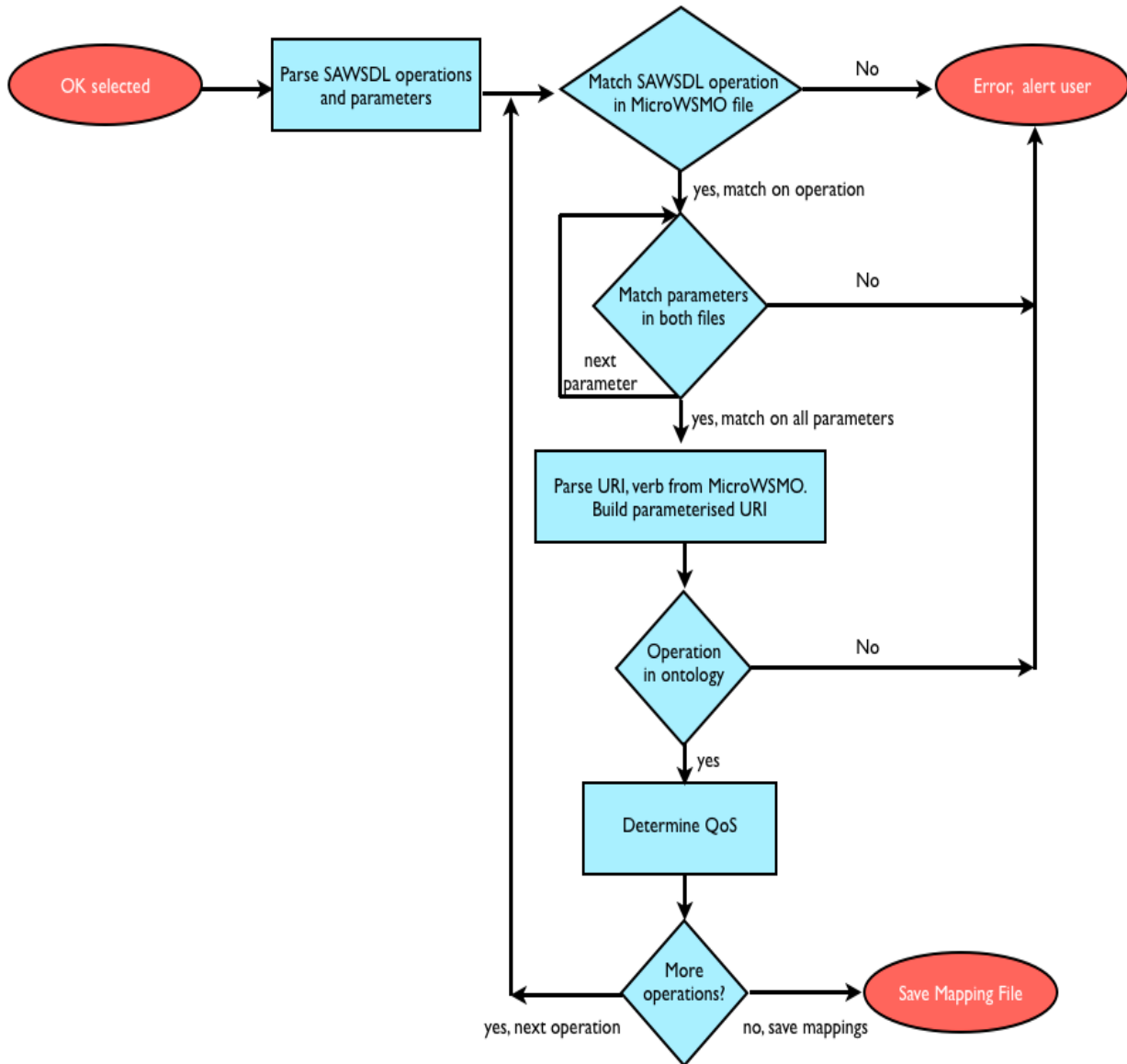
4.5. Example

In this section an example detailing the semantic matching/automation is given. Sample file segments relating to Figure 6 are presented. The example will explain why, as shown in Figure 9, there is no lifting/lowering necessary in StoRHm.

4.5.1. SAWSDL Example

Listing 1 is a sample SAWSDL file outlining a SOAP Banking Web Service. Note that a SAWSDL file is a WSDL file with semantic annotations (as outlined in Figure 6).

Listing 1. SAWSDL example (segment).

```
30  <xsd:complexType name="ServiceView">
31    <xsd:sequence>
32      <xsd:element minOccurs="0" name="branchCode"
33              sawsdl:modelReference="http://www.leitrimmills.ie/ontologies/BankService#BranchCode"
34              type="xsd:string"/>
35      <xsd:element minOccurs="0" name="accountNo"
36              sawsdl:modelReference="http://www.leitrimmills.ie/ontologies/BankService#AccountNo"
37              type="xsd:string"/>
38    </xsd:sequence>
39  </xsd:complexType>
40  </xsd:schema>
41 </types>
42 <message name="ServiceView">
43   <part element="tns:ServiceView" name="parameters"/>
44 </message>
45
46 <portType name="BankService">
47   <operation name="ServiceView">
48     <input message="tns:ServiceView"/>
49     <output message="tns:ServiceViewResponse"/>
50     <sawsdl:attrExtensions sawsdl:modelReference="http://www.leitrimmills.ie/ontologies/BankService#ViewService"/>
51   </operation>
52 </portType>
```

Note the semantic annotations of the operation and its input parameters:
"*http://www.leitrimmills.ie/ontologies/BankService#ViewService*" (line 50, the operation);
"*http://www.leitrimmills.ie/ontologies/BankService#BranchCode*" (line 33, first input parameter);
"*http://www.leitrimmills.ie/ontologies/BankService#AccountNo*" (line 36, second input parameter).

4.5.2. MicroWSMO/hRESTS Example

Listing 2 shows the MicroWSMO/hRESTS equivalent of Listing 1. The hRESTS annotations enable machine automation and are easily identified on lines 9 and 10 with the *hr* prefix. In addition, hRESTS leverages an existing model, the Minimal Service Model (*msm* prefix) used across various ontologies in the Core Dashboard suite. Given the similarity between MicroWSMO and SAWSDL, it is not surprising to note the *sawsdl* namespace prefix for MicroWSMO semantic annotation on lines 13, 18 and 23 of Listing 2. These three lines correspond to the RESTful Web service operation and its associated input parameters respectively:

"*http://www.leitrimmills.ie/ontologies/BankService#ViewService*" (line 13, the operation)
"*http://www.leitrimmills.ie/ontologies/BankService#BranchCode*" (line 18, first input parameter)
"*http://www.leitrimmills.ie/ontologies/BankService#AccountNo*" (line 23, second input parameter)

Note that the semantic concepts, identified by URI references, match between Listings 1 and 2. This is how the matching of the SOAP WS to its RESTful counterpart and the resultant automation is achieved. The MicroWSMO file provides the HTTP verb and URI used by the equivalent RESTful Web Service (see lines 9 and 11 respectively of Listing 2).

Listing 2. MicroWSMO example (segment).

```
3   <msm:Service rdf:ID="BankService">
4     <rdfs:isDefinedBy rdf:resource=""/>
5     <rdfs:label>Bank Service API</rdfs:label>
6     <msm:hasOperation>
7       <msm:Operation rdf:ID="ViewOperation">
8         <rdfs:label>ServiceView</rdfs:label>
9         <hr:hasMethod>GET</hr:hasMethod>
10        <hr:hasAddress rdf:datatype="http://www.wsmo.org/ns/hrests#URITemplate">
11          http://127.0.0.1:3051/RESTServer/BankServices/{branchcode}/{accountnumber}
12        </hr:hasAddress>
13        <sawsdl:modelReference rdf:resource="http://www.leitrimmills.ie/ontologies/BankService#ViewService"/>
14        <msm:hasInput>
15          <msm:MessageContent rdf:ID="ViewOperationInput">
16            <msm:hasOptionalPart>
17              <msm:MessagePart rdf:ID="branchcode">
18                <sawsdl:modelReference rdf:resource="http://www.leitrimmills.ie/ontologies/BankService#BranchCode"/>
19              </msm:MessagePart>
20            </msm:hasOptionalPart>
21            <msm:hasOptionalPart>
22              <msm:MessagePart rdf:ID="accountnumber">
23                <sawsdl:modelReference rdf:resource="http://www.leitrimmills.ie/ontologies/BankService#AccountNo"/>
24              </msm:MessagePart>
25            </msm:hasOptionalPart>
26          </msm:MessageContent>
27        </msm:hasInput>
28      </msm:Operation>
```

4.5.3. Ontology Example

To determine the QoS required, the service is located in the ontology. Listing 3 is the ontology for the service described in Listings 1 and 2. The *ViewService* concept is outlined between lines 73–77. The URIRef "*http://www.leitrimmills.ie/ontologies/BankService#ViewService*" (line 73) denotes the semantic concept. This is the same semantic concept identified on line 50 of Listing 1 (SAWSDL file) and on line 13 of Listing 2 (MicroWSMO file). The ontology states that this service does not support Security or Reliability (lines 75–76).

Listing 3. Ontology example (segment).

```
71    <!-- http://www.leitrimmills.ie/ontologies/BankService#ViewService -->
72
73    <BankService rdf:about="#ViewService">
74        <rdf:type rdf:resource="&owl;Thing"/>
75        <hasReliability rdf:datatype="&xsd;boolean">false</hasReliability>
76        <hasSecurity rdf:datatype="&xsd;boolean">false</hasSecurity>
77    </BankService>
```

Once the QoS for the service has been ascertained, the CSV file can be populated for that service. The CSV file is populated as follows:
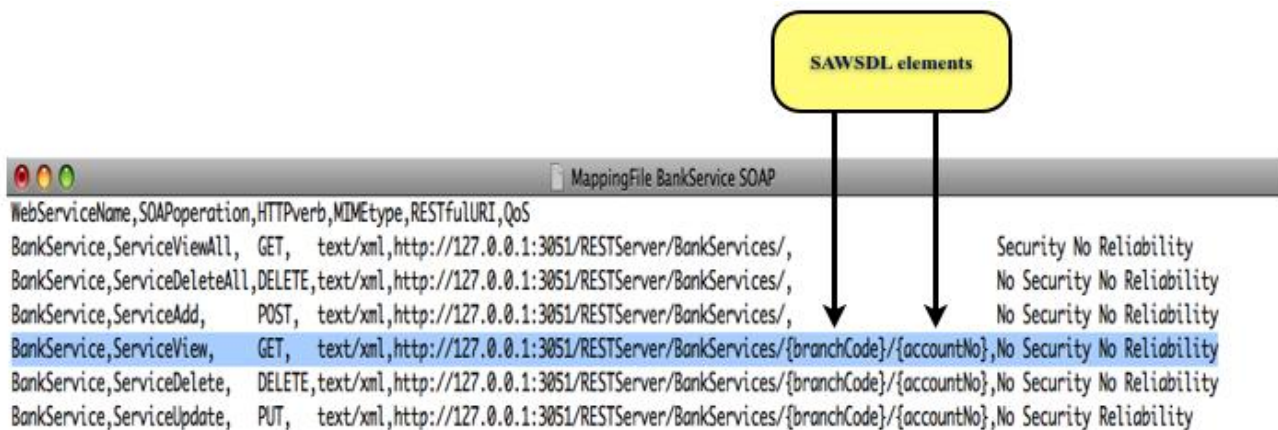
- Web Service Name and SOAP Operation from the SAWSDL file
- RESTful URI and HTTP verb from the MicroWSMO file
- QoS from the ontology

However, if any of the SAWSDL concepts are not located in either the MicroWSMO and/or ontology files, an error is reported to the user and the configuration process exits without creating the CSV file.

4.5.4. CSV Example

The highlighted entry in Listing 4 below shows the CSV file created based on the annotations outlined:

Listing 4. CSV sample file.

4.5.5. No Lifting/Lowering Necessary

Note that the RESTful URI of Listing 4 (column 5) in the highlighted entry contains the SOAP message/element names from the SAWSDL file *i.e.*, *branchCode* and *accountNo* (lines 32 and 35 in Listing 1 respectively). If the exiting URI from the MicroWSMO file had been used, the RESTful URI in Listing 4 would have ended "*/{branchcode}/{accountnumber}*" (line 11 in Listing 2). Though the difference in case is slight, the impact is significant. *By inserting the SAWSDL elements into the RESTful URI, the StoRHm configurator is effectively inserting XPath expressions that enable the StoRHm adapter to parse the SOAP message.* This was a deliberate design decision and impacts on the requirement for lifting/lowering.

The confluence of two factors results in the fact that there is no need for semantic lifting/lowering in StoRHm:

a) SOAP is a standard message format and thus its structure is well known; this enables the automatic parsing of the SOAP message as far as the SOAP *Body* element.

b) the remaining XPath expressions required to parse the SOAP *Body* element are obtained from the RESTful URI of the CSV file.

Thus, with the semantic concepts of MicroWSMO informing StoRHm *where* to insert the data in the RESTful URI; and the CSV, with its in-built XPath expressions based on SOAP element names, informing StoRHm *what* data to insert; StoRHm is able to transform the SOAP message to RESTful format without the need for lifting/lowering. This means that the ontology is used for matching operations and determining the QoS associated with the relevant operations.

4.6. Research Methodology

The research strategy used for the software artefacts was design and creation. Experimentation was the strategy used for testing and performance measurement of the framework. The data generation method is document based *i.e.*, the generated CSV file, SOAP and RESTful HTTP messages were examined as part of testing. Lastly, the data analysis method is qualitative.

5. Testing

StoRHm v2 was tested in an Internet based setting (as opposed to StoRHm v1 which was tested on a local machine). In this section, the test environment is outlined and performance of the adapter is analysed.

5.1. Test Environment

A test environment has been implemented in an Internet based setting (see Figure 10). The client machine is a Dell Optiplex 780 desktop machine running Windows 7 with 4GB RAM and Intel Core2 Duo CPU processors. The server is a Dell Dimension 8400 running Windows XP with 3GB RAM. One of the Dell machines hosts the client and protocol adapter software while the other machine hosts the server software and database. Glassfish [22] is the application server used. The server machine is connected to the Internet via a 3Mb/sec ADSL line.

Figure 10. Test environment.
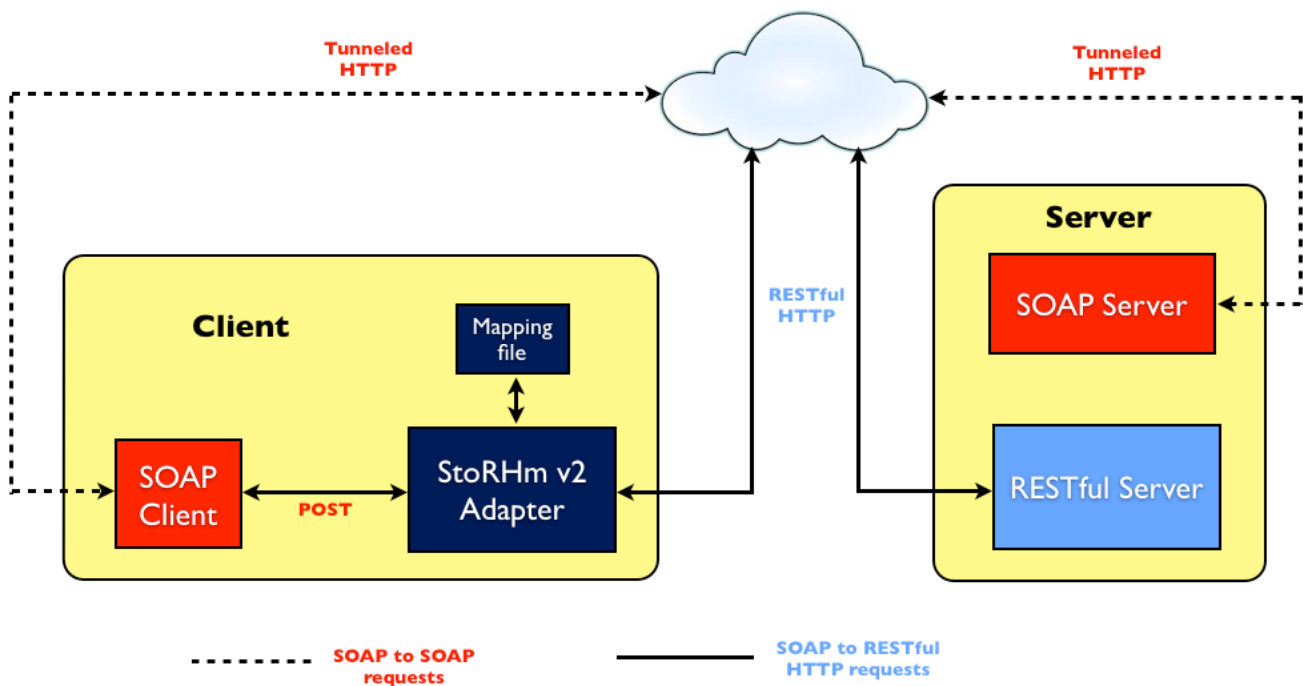
5.2. Test Results

While performance is not central to this research, performance is an important consideration for any distributed technology and therefore the latency delay due to the introduction of the adapter was

measured. One of the standard measurements taken is the "*round-trip method invocation time*" which is the time difference between the client invoking the remote method and the remote method returning its results [23].

In Table 1, we present performance tests comparing the round-trip method invocation times (in msecs) of a normal SOAP WS request with that of the same SOAP request routed via the adapter to its equivalent RESTful WS. The Web Services were deliberately chosen so as to map to different HTTP verbs: *ServiceView (GET)*; *ServiceAdd (POST)*; *ServiceDelete (DELETE)* and *ServiceUpdate (PUT)*. Note that the adapter-based figures represent a worst-case scenario *i.e.*, a full round trip on each occasion with no efficiencies such as caching implemented.

Table 1. Internet Adapter statistics.

SOAP Statistics (all figures in msecs)		Average	Standard Deviation	Standard Error	zLow 95%	zHigh 95%	Verb	Adapter +/–	Significant Value 95% (+/– **1.645**)
ServiceView	no adapter	67	12.08	1.2	65	69	POST	n/a	3.45
	adapter	72.11	8.54	0.85	70	74	GET	+8%	
ServiceAdd	no adapter	65.29	7.95	0.79	64	67	POST	n/a	2.23
	adapter	68.98	14.48	1.44	66	72	POST	6%	
ServiceDelete	no adapter	66.87	13.64	1.36	64	70	POST	n/a	3.39
	adapter	72.22	7.85	0.78	71	74	DELETE	+8%	
ServiceUpdate	no adapter	65.46	7.65	0.76	64	67	POST	n/a	2.67
	adapter	67.84	4.52	0.45	67	69	PUT	+4%	

Each SOAP WS was directly accessed (no adapter) 100 times in order to get an average delay. In addition, the following statistical values were also recorded: the standard deviation, standard error and low and high values (with 95% confidence). The tests were then repeated 100 times with the client accessing the RESTful Web Services via the adapter and the results recorded again.

In situations where the adapter maps to PUT or POST, the adapter performs **no** parsing and the entity body (of the RESTful HTTP request) contains the original SOAP message request. As Table 1 shows, there is a 4–8% penalty for inserting the adapter into an infrastructure.

5.3. Statistical Analysis of Results

Table 1–In this scenario, a one-tailed test is appropriate. At the 95% significance level, the t-table value is 1.645. All rows are applicable in Table 1. *ServiceView*, *ServiceAdd*, *ServiceDelete* and *ServiceUpdate* have test statistic values of 3.45, 2.23, 3.39 and 2.67 respectively. As all values are greater than 1.645, the alternative hypothesis is therefore accepted *i.e.*, the average time taken to route

these SOAP requests via the adapter to their RESTful Web Service equivalents is significantly longer (statistically) than the average time taken to issue these requests directly to the SOAP Server.

*5.4. Linear Regression Analysis*

The adapter behaves consistently when mapping to PUT/POST *i.e.*, the message body is copied from the incoming SOAP request to the outgoing RESTful request. However, this is not the case with GET/DELETE, where parsing of the incoming SOAP message is required to populate the RESTful URI. The example SOAP files used in the requests to create Table 1 were based on sample files used in industry. All the files were 1643 bytes in size. In addition, these files required the adapter to perform very little parsing when mapping to a GET or DELETE (the adapter had to evaluate 2 XPath expressions). Therefore, further tests were required and undertaken in order to measure whether or not the penalty imposed by the adapter when mapping to GET/DELETE is a function of:

  a) the size of the SOAP request file and/or
  b) the amount of parsing to be conducted by the adapter

Size of SOAP request file

Figure 11 below represents test results where the request file size is increased steadily for 15 iterations in an environment where there is no adapter in situ *i.e.*, the requests travel directly to the server. All response times are based of averages of 100 requests.

Figure 11. Performance–increasing file sizes, static parsing, no adapter.
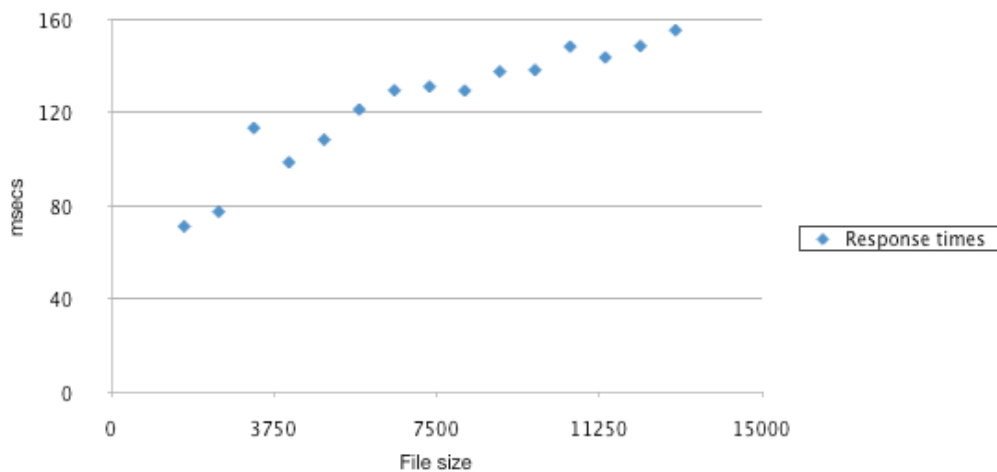
As Figure 11 demonstrates, the response times increase as and when the request file sizes increase. The correlation value between the size of the request files and the response times is 0.936. Thus, as one would expect, when requests travel directly to the server, there is a strong correlation between the size of the request files and the response times.

Figure 12 represents the graph where the exact same requests are issued with the adapter in situ *i.e.*, the requests travel to the server via the adapter. The requests map to a GET and in order to isolate the file size impact, the parsing requirement remains constant across the requests (2 XPath expressions to be evaluated as in Table 1).

Figure 12. Performance–increasing file sizes, static parsing, via adapter.
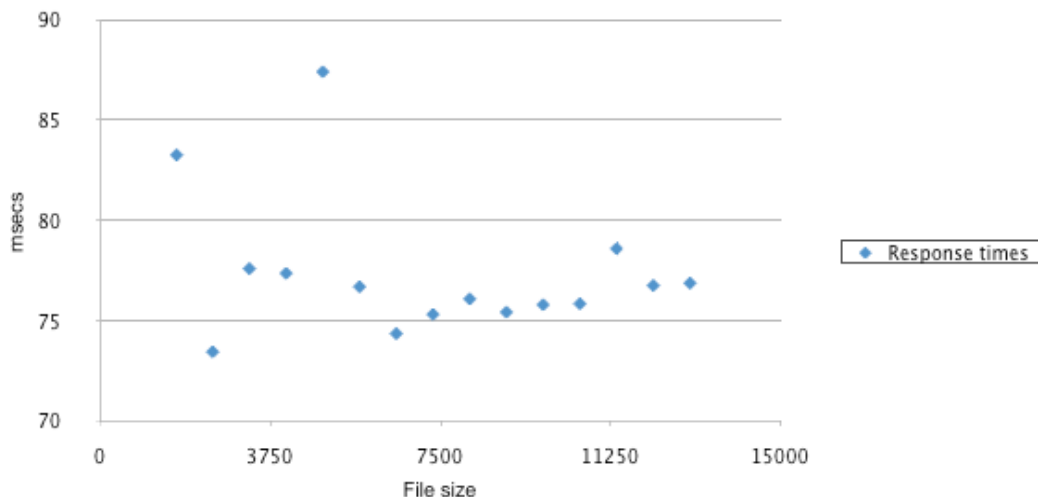
The correlation value between the size of the request file and the response time when the adapter is in place is −0.276. Thus, there is a very weak linear relationship between the size of the request file and the response time taken when requests travel via the adapter. If we fitted the least squares line to the data the equation would be:

$$\text{Time} = -3/10000 \times \text{file size} + 79.35$$

The extremely weak correlation value is demonstrated by the linear regression equation. The response time is almost entirely dependent on the constant. Thus, the file size has virtually no bearing on the response times. Two factors explain this:

a) Firstly, the adapter resides on the local machine and thus the increase in file size is not as influential as it would be if the adapter resided on a remote machine

b) Secondly and more importantly, the parsing conducted by the adapter is consistent across all the requests, regardless of file size. This means that, regardless of the increasing file sizes arriving at the adapter, the subsequent RESTful requests emanating from the adapter *are all the same size*. It is interesting to note by comparing Figure 12 with Figure 11 that the penalty imposed by the adapter disappears as the file size increases. The data in Table 1 was generated using a sample industry file of size 1643 bytes. At that file size, the adapter does impose a statistically significant penalty.

Point (b) above is best explained with an example. Listing 5 is a sample CSV file segment used by the adapter to map SOAP requests to RESTful HTTP format. Listing 5 states that the SOAP operation *ServiceView* in the Web Service *BankService* is mapped to a GET on the URI *http://127.0.0.1:3050/RESTServer/BankServices*. The XPath expressions {*branchCode*} and {*accountNo*} are used by the adapter to parse these elements from the incoming SOAP message and insert them into the RESTful GET request.

Listing 5. Sample CSV file segment.

```
         WebServiceName,    SOAPoperation, HTTPverb,    MIMEtype,
RESTfulURI,                                                                  QoS
         BankService,       ServiceView,    GET,       text/xml,

http://127.0.0.1:3050/RESTServer/BankServices/{branchCode}/{accountNo}  , No Security No Reliability
```

Listing 6 is the sample *ServiceView* file used in generating Table 1. On receiving the request in Listing 6, the adapter, informed by the mapping file in Listing 5 generates an HTTP GET request on the URI *http://127.0.0.1:3050/RESTServer/BankServices/123456/12345678.*

Listing 6. Sample request file that maps to GET.

```xml
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
     <ns2:ServiceView xmlns:ns2="http://BankServicePkg/">
               <branchCode>123456</branchCode>
               <accountNo>12345678</accountNo>
               <CUSTOMER_TITLE>Mr.</CUSTOMER_TITLE>
               <CUSTOMER_FIRSTNAME>Joseph</CUSTOMER_FIRSTNAME>
               <CUSTOMER_MIDDLE_INITIAL>P</CUSTOMER_MIDDLE_INITIAL>
               <CUSTOMER_SURNAME>Bloggs</CUSTOMER_SURNAME>
               <CUSTOMER_ADDRESS_LINE1>10 Main Street</CUSTOMER_ADDRESS_LINE1>
               <CUSTOMER_ADDRESS_LINE2>Athlone</CUSTOMER_ADDRESS_LINE2>
               <CUSTOMER_ADDRESS_LINE3>Co. Westmeath</CUSTOMER_ADDRESS_LINE3>
               <CUSTOMER_ADDRESS_LINE4>Ireland</CUSTOMER_ADDRESS_LINE4>
               <CUSTOMER_TYPE>Personal</CUSTOMER_TYPE>
               <CUSTOMER_RATING>4</CUSTOMER_RATING>
               <COMMENT_LINE1>This is a comment with regard to the above customer...</COMMENT_LINE1>
               <COMMENT_LINE2>This is a comment with regard to the above customer...</COMMENT_LINE2>
               <COMMENT_LINE3>This is a comment with regard to the above customer...</COMMENT_LINE3>
               <COMMENT_LINE4>This is a comment with regard to the above customer...</COMMENT_LINE4>
               <COMMENT_LINE5>This is a comment with regard to the above customer...</COMMENT_LINE5>
               <COMMENT_LINE6>This is a comment with regard to the above customer...</COMMENT_LINE6>
               <COMMENT_LINE7>This is a comment with regard to the above customer...</COMMENT_LINE7>
               <COMMENT_LINE8>This is a comment with regard to the above customer...</COMMENT_LINE8>
               <COMMENT_LINE9>This is a comment with regard to the above customer...</COMMENT_LINE9>
     </ns2:ServiceView>
</S:Body>
</S:Envelope>
```

Listing 7 is a larger *ServiceView* file. The important point here is that, Listing 7 generates the same HTTP GET request even though it is a much larger file–Listing 6 is 1643 bytes in size and Listing 7 is 13,000 bytes in size.

Listing 7. Larger sample request file that maps to GET.

```xml
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
      <ns2:ServiceView xmlns:ns2="http://BankServicePkg/">
                <branchCode>123456</branchCode>
                <accountNo>12345678</accountNo>
                <CUSTOMER_TITLE>Mr.</CUSTOMER_TITLE>
                <CUSTOMER_FIRSTNAME>Joseph</CUSTOMER_FIRSTNAME>
                <CUSTOMER_MIDDLE_INITIAL>P</CUSTOMER_MIDDLE_INITIAL>
                <CUSTOMER_SURNAME>Bloggs</CUSTOMER_SURNAME>
                <CUSTOMER_ADDRESS_LINE1>10 Main Street</CUSTOMER_ADDRESS_LINE1>
                <CUSTOMER_ADDRESS_LINE2>Athlone</CUSTOMER_ADDRESS_LINE2>
                <CUSTOMER_ADDRESS_LINE3>Co. Westmeath</CUSTOMER_ADDRESS_LINE3>
                <CUSTOMER_ADDRESS_LINE4>Ireland</CUSTOMER_ADDRESS_LINE4>
                <CUSTOMER_TYPE>Personal</CUSTOMER_TYPE>
                <CUSTOMER_RATING>4</CUSTOMER_RATING>
                <COMMENT_LINE1>This is a comment with regard to the above customer...</COMMENT_LINE1>
                <COMMENT_LINE2>This is a comment with regard to the above customer...</COMMENT_LINE2>
                <COMMENT_LINE3>This is a comment with regard to the above customer...</COMMENT_LINE3>
                <COMMENT_LINE4>This is a comment with regard to the above customer...</COMMENT_LINE4>
                <COMMENT_LINE5>This is a comment with regard to the above customer...</COMMENT_LINE5>
                <COMMENT_LINE6>This is a comment with regard to the above customer...</COMMENT_LINE6>
                <COMMENT_LINE7>This is a comment with regard to the above customer...</COMMENT_LINE7>
                <COMMENT_LINE8>This is a comment with regard to the above customer...</COMMENT_LINE8>
                <COMMENT_LINE9>This is a comment with regard to the above customer...</COMMENT_LINE9>
                <COMMENT_LINE1>This is a comment with regard to the above customer...</COMMENT_LINE1>
                <COMMENT_LINE1>This is a comment with regard to the above customer...</COMMENT_LINE1>
                <COMMENT_LINE1>This is a comment with regard to the above customer...</COMMENT_LINE1>
                <COMMENT_LINE1>This is a comment with regard to the above customer...</COMMENT_LINE1>
                <COMMENT_LINE1>This is a comment with regard to the above customer...</COMMENT_LINE1>
                <COMMENT_LINE1>This is a comment with regard to the above customer...</COMMENT_LINE1>
                    120 lines like this...
      </ns2:ServiceView>
</S:Body>
</S:Envelope>
```
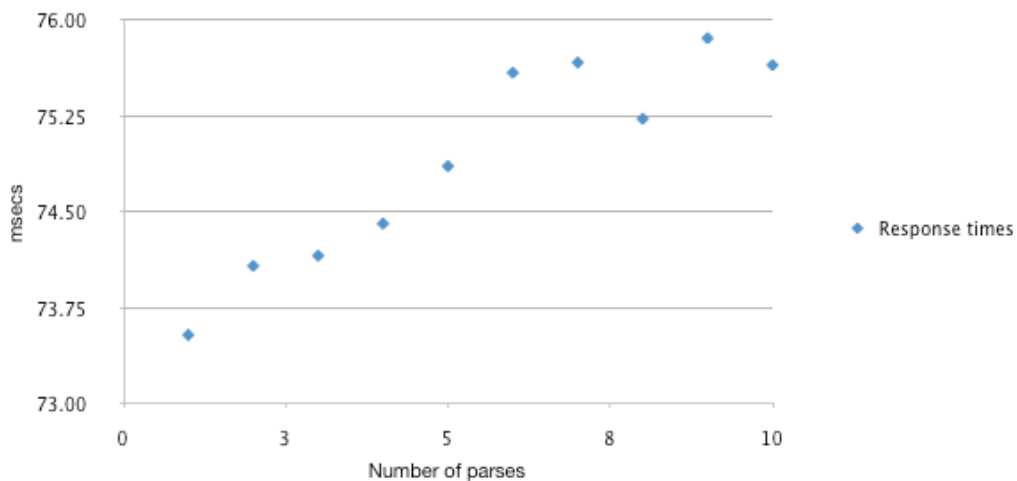
Parsing requirement

In order to affirm that the penalty imposed by the adapter is a function of the parsing requirement, testing was performed whereby the file size remained constant (the file from Listing 6 was repeatedly used) but the parsing requirement increased. Figure 13 represents the results.

Figure 13. Performance–increasing parsing requirement, static file sizes, via adapter.

As is evident from the graph, the adapter penalty increases in line with the parsing requirement. The correlation value of 0.931 supports this assertion. Thus, there is a strong positive correlation between

the parsing requirement of the adapter and the response times: the greater the number of XPath expressions to be evaluated by the adapter, the higher the method invocation response times.

As stated previously, the authors based their testing on a sample industry file. This file has limited nesting and the evaluation of 10 XPath expressions was more than sufficient. Listing 8 below shows the XPath expressions (enclosed by {}), evaluated by the adapter in the generation of the results for Figure 13. They are listed in order, from 1 to 10, delimited by "/". For example, {*branchCode*} requires 1 XPath expression to be evaluated whereas {*branchCode*}/{*accountNo*} requires 2 XPath expressions to be evaluated.

Listing 8. XPath expressions evaluated.

```
{branchCode}

{branchCode}/{accountNo}

{branchCode}/{accountNo}/{CUSTOMER_TITLE}

{branchCode}/{accountNo}/{CUSTOMER_TITLE}/{CUSTOMER_FIRSTNAME}

{branchCode}/{accountNo}/{CUSTOMER_TITLE}/{CUSTOMER_FIRSTNAME}/{CUSTOMER_MIDDLE_INITIAL}

{branchCode}/{accountNo}/{CUSTOMER_TITLE}/{CUSTOMER_FIRSTNAME}/{CUSTOMER_MIDDLE_INITIAL}/{CUSTOMER_SURNAME}

{branchCode}/{accountNo}/{CUSTOMER_TITLE}/{CUSTOMER_FIRSTNAME}/{CUSTOMER_MIDDLE_INITIAL}/{CUSTOMER_SURNAME}/
{CUSTOMER_ADDRESS_LINE1}

{branchCode}/{accountNo}/{CUSTOMER_TITLE}/{CUSTOMER_FIRSTNAME}/{CUSTOMER_MIDDLE_INITIAL}/{CUSTOMER_SURNAME}/
{CUSTOMER_ADDRESS_LINE1}/{CUSTOMER_ADDRESS_LINE2}

{branchCode}/{accountNo}/{CUSTOMER_TITLE}/{CUSTOMER_FIRSTNAME}/{CUSTOMER_MIDDLE_INITIAL}/{CUSTOMER_SURNAME}/
{CUSTOMER_ADDRESS_LINE1}/{CUSTOMER_ADDRESS_LINE2}/{CUSTOMER_ADDRESS_LINE3}

{branchCode}/{accountNo}/{CUSTOMER_TITLE}/{CUSTOMER_FIRSTNAME}/{CUSTOMER_MIDDLE_INITIAL}/{CUSTOMER_SURNAME}/
{CUSTOMER_ADDRESS_LINE1}/{CUSTOMER_ADDRESS_LINE2}/{CUSTOMER_ADDRESS_LINE3}/{CUSTOMER_ADDRESS_LINE4}
```

Note that the structure of the SOAP message is enterprise-specific which could result in various different levels of nesting within the SOAP message. As a result, we performed extra tests with a different level of nesting. Figure 14 represents the results.

Figure 14. Performance–increased nesting, static file sizes, via adapter.
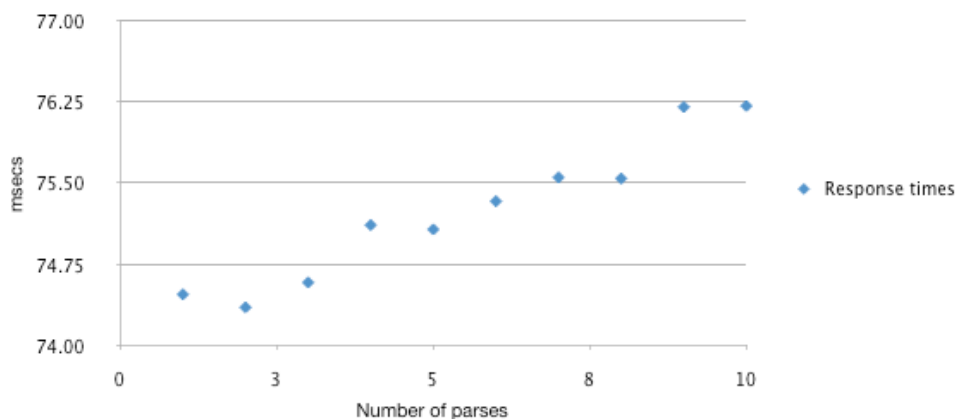
The correlation value in this instance is 0.973. Thus, the strong positive correlation between the amount of parsing to be performed by the adapter and the response times is once again confirmed. In fact, with the extra level of nesting, the correlation is slightly stronger.

Listing 9 below shows the XPath expressions evaluated by the adapter in the generation of the results for Figure 14. They are listed in order, from 1 to 10, delimited by "/". The extra level is encapsulated by the element *inputData*.

Listing 9. XPath expressions evaluated.

```
{inputData/branchCode}

{inputData/branchCode}/{inputData/accountNo}

{inputData/branchCode}/{inputData/accountNo}/{inputData/CUSTOMER_TITLE}

{inputData/branchCode}/{inputData/accountNo}/{inputData/CUSTOMER_TITLE}/{inputData/CUSTOMER_FIRSTNAME}

{inputData/branchCode}/{inputData/accountNo}/{inputData/CUSTOMER_TITLE}/{inputData/CUSTOMER_FIRSTNAME}/{inputData/
CUSTOMER_MIDDLE_INITIAL}

{inputData/branchCode}/{inputData/accountNo}/{inputData/CUSTOMER_TITLE}/{inputData/CUSTOMER_FIRSTNAME}/{inputData/
CUSTOMER_MIDDLE_INITIAL}/{inputData/CUSTOMER_SURNAME}

{inputData/branchCode}/{inputData/accountNo}/{inputData/CUSTOMER_TITLE}/{inputData/CUSTOMER_FIRSTNAME}/{inputData/
CUSTOMER_MIDDLE_INITIAL}/{inputData/CUSTOMER_SURNAME}/{inputData/CUSTOMER_ADDRESS_LINE1}

{inputData/branchCode}/{inputData/accountNo}/{inputData/CUSTOMER_TITLE}/{inputData/CUSTOMER_FIRSTNAME}/{inputData/
CUSTOMER_MIDDLE_INITIAL}/{inputData/CUSTOMER_SURNAME}/{inputData/CUSTOMER_ADDRESS_LINE1}/{inputData/
CUSTOMER_ADDRESS_LINE2}

{inputData/branchCode}/{inputData/accountNo}/{inputData/CUSTOMER_TITLE}/{inputData/CUSTOMER_FIRSTNAME}/{inputData/
CUSTOMER_MIDDLE_INITIAL}/{inputData/CUSTOMER_SURNAME}/{inputData/CUSTOMER_ADDRESS_LINE1}/{inputData/
CUSTOMER_ADDRESS_LINE2}/{inputData/CUSTOMER_ADDRESS_LINE3}

{inputData/branchCode}/{inputData/accountNo}/{inputData/CUSTOMER_TITLE}/{inputData/CUSTOMER_FIRSTNAME}/{inputData/
CUSTOMER_MIDDLE_INITIAL}/{inputData/CUSTOMER_SURNAME}/{inputData/CUSTOMER_ADDRESS_LINE1}/{inputData/
CUSTOMER_ADDRESS_LINE2}/{inputData/CUSTOMER_ADDRESS_LINE3}/{inputData/CUSTOMER_ADDRESS_LINE4}
```

6. Evaluation

Our hypothesis was that SWS technologies could be leveraged to automate the previously manual configuration element of StoRHm v1. In addition, Internet based latency tests of the protocol adapter element were required.

6.1. Configuration Wizard

The wizard prompts the user for the names of the SAWSDL, MicroWSMO and ontology files. The user selects OK. The SOAP operations and associated input parameters' model references will be parsed from the SAWSDL file. The equivalent concepts will be located in the MicroWSMO and ontology files using the model references. Data from all three input files is then used to populate the CSV file.

6.2. Protocol Adapter

There is a 4–8% penalty for inserting the adapter into an infrastructure. Statistically speaking, the average time taken to route these SOAP requests via the adapter to their RESTful Web Service equivalents is significantly longer than the average time taken to issue these requests directly to the SOAP Server. Performance measurements demonstrated that increasing the size of the request files while maintaining a constant parsing requirement does not impact negatively. Conversely, increasing the parsing requirement across static file sizes, does impact performance.

7. Conclusions and Future Work

We have implemented, tested and demonstrated an automated configuration wizard. The configuration wizard is automated in line with SWS technologies best practice. The protocol adapter has been tested in an Internet based setting. There is a 4–8% time penalty involved in using the adapter which is statistically significant. This time penalty is dependent on the amount of parsing the adapter must perform.

The following areas outline where we wish to focus next:

- One of the constraints imposed by the architecture is that, in situations where the target HTTP verb to be used is PUT or POST, the entity body of the request is sent on untouched. Typically, the outer element of the SOAP Body element contains the operation to be executed e.g., the WSDL wrapped document-literal pattern enforces this. However, this "operation" element is not needed by RESTful HTTP implementations as the "operation" is identified by the URI coupled with the verb. In order to address this, research is required on XSLT transformations to cater for scenarios as described above, where the XML content to be passed on differs from the XML content received.

- This paper is not centered on performance and consequently the performance tests carried out are indicative rather than extensive. The latency performance of the adapter could be extended. This would include: the performance impact of message reliability and the effect of multiple similar requests with efficiencies such as caching and Conditional GET in place.

- The architecture enables SOAP clients to access pre-existing RESTful HTTP Web Services. The adapter is a client-side migration enabler. Research could be conducted to extend the framework to focus on the server *i.e.*, provide a server-side migration enabler from SOAP WS to RESTful WS. Should an enterprise wish to migrate from SOAP WS to RESTful WS, this new extension would be executed first to migrate the server. With the server migrated, the current framework would then be used to enable the enterprise to gradually migrate the clients.

References

1. Resource Description Framework specification. Available online: http://www.w3.org/RDF/ (accessed on 5 April 2012).
2. Hebeler, J.; Fisher, M.; Blace, R.; Perez-Lopez, A. *Semantic Web Programming*; Wiley: Indianapolis, IN, USA, 2009.
3. RDF Schema specification. Available online: http://www.w3.org/TR/rdf-schema/ (accessed on 5 April 2012).
4. Web Ontology Language (OWL) specification. Available online: http://www.w3.org/ TR/owl-semantics/ (accessed on 5 April 2012).
5. Gruber, T. Ontology. In *Encyclopedia of Database Systems*; Liu, L., Özsu, M.T., Eds.; Springer-Verlag: Berlin, Germany, 2008.
6. Kennedy, S.; Molloy, O.; Stewart, R.; Jacob, P. StoRHm: A protocol adapter for mapping SOAP based Web Services to RESTful HTTP format. *Electron. Commer. Res. J.* **2011**, *11*, 245–269.

7.   Battle, R.; Benson, E. Bridging the Semantic Web with Representational State Transfer (REST). *J. Web Semant.* **2008**, *6*, 61–69.

8.   OWL-S: Semantic Markup for Web Services. Available online: http://www.w3.org/ Submission/OWL-S (accessed on 5 April 2012).

9.   SAWSDL: Semantic Annotations for WSDL and XML Schema. Available online: http://www.w3.org/TR/sawsdl/ (accessed on 5 April 2012).

10. Lathem, J.; Gomadam, K.; Sheth, A. SA-REST and (S)mashups: Adding Semantics to RESTful Services. In *Proceedings of International Conference on Semantic Computing*, Irvine, CA, USA, 17–19 September 2007; pp. 469–476.

11. Kopecky, J.; Vitvar, T.; Fensel, D. MicroWSMO and hRESTS, Technical Report 2009. Available online: http://sweet.kmi.open.ac.uk/pub/microWSMO.pdf (accessed on 5 April 2012).

12. Gleaning Resource Descriptions from Dialects of Languages (GRDDL) specification. Available online: http://www.w3.org/TR/grddl/ (accessed on 5 April 2012).

13. RDF in Attributes (RDFa) specification. Available online: http://www.w3.org/TR/ rdfa-syntax/ (accessed on 5 April 2012).

14. SOA4All Core Dashboard tool suite. Available online: http://coconut.tie.nl:8080/ dashboard/#1304859734132 (accessed on 5 April 2012).

15. Yahoo! Pipes. Available online: http://pipes.yahoo.com/pipes/ (accessed on 5 April 2012).

16. Sycara, K.; Pauoucci, M.; Ankolekar, A.; Srinivasan, N. Automated discovery, interaction and composition of Semantic Web Services. *J. Web Semant.* **2003**, *1*, 27–46.

17. Fensel, D.; Fischer, F.; Kopecky, J.; Krummenacher, R.; Lambert, D.; Vitvar, T. WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. Available online: http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/ (accessed on 5 April 2012).

18. Bizer, C.; Heath, T.; Berners-Lee, T. Linked Data – The Story So Far. *Int. J. Semant. Web In. Syst.* **2009**, *5*, 1–22.

19. Paliwal, A.; Shafiq, B.; Vaidya, J.; Ziong, H.; Adam, N. Semantics based automated service discovery. *IEEE Trans. Serv. Comput*. **2011**, *99*, 1.

20. Maleshkova, M.; Pedrinaci, C.; Domingue, J. Semantic annotation of Web APIs with SWEET. In *Proceedings of 6th Workshop on Scripting and Development for the Semantic Web at Extended Semantic Web Conference*, Crete, Greece, 31 May 2010.

21. Protégé. Available online: http://protege.stanford.edu/ (accessed on 5 April 2012).

22. Glassfish Open Source Application Server. Available online: http://glassfish.java.net/ (accessed on 5 April 2012).

23. Juric, M.; Kezmah, B.; Hericko, M.; Rozman, I.; Vezocnik, I. Java RMI, RMI Tunneling and Web Services Comparison and Performance Analysis. *ACM SIGPLAN Notices* **2004**, *39*, 58–65.