Appendix A
Translation of Heterogeneous Service Descriptions to Proteus Service Advertisements

One of the most valuable features of the Proteus query model is its ability to operate in diverse service-oriented environments, where services with different characteristics are described by heterogeneous languages. Such ability is attained by enabling the proper translation of the heterogeneous service descriptions into a common format, which is conveniently provided by the Proteus service advertisement. It is important to stress that, the latter is not intended as an alternative solution to service description. Rather, it provides Proteus with a convenient, unified way of describing the various service properties, so that service discovery queries can be formulated and executed in a service technology-independent manner. In this appendix, we present the algorithms and entailed mapping rules that drive the translation of documents written in various service description languages into Proteus service advertisements, while also ensuring that no useful information is lost in the process.

A.1 Translating WSDL Documents

The Web Services Description Language (WSDL)[1] is the predominant industry standard for the description of Web services. A WSDL document is fundamentally split into two parts: The *abstract* part is used to describe the interface of a Web service, in terms of the operations it provides, and the messages that these operations consume and/or produce. The *concrete* part on the other hand is used to describe the implementation of that interface, in terms of the technology-specific bindings and ports required to access and invoke these operations.

Even though service requesters commonly set their requirements on the capability, input, and output of the desired Web service operation, which are described by the abstract part of the WSDL document, it is important that the Proteus service advertisement also captures the invocation details of the concrete part. This necessity is accommodated by our Algorithm A.1, which illustrates the way the contents of a WSDL document are traversed upon translation. As it can be seen in lines 9–28, the translator generates one service advertisement for each operation found in the document that also has an implementation, i.e. it is reachable through a port. Thus, each produced service advertisement corresponds to a Web service operation and is generally populated with five properties, namely the *Service*, *Capability*, *Input*, *Output*, and *WSInvocationInfo*, which result from the application of the following mapping rules.

The wsdl:service element is mapped to the Service property (see line 11). The latter uses the *string value* feature to capture the name attribute of that element, while the *semantics* feature is employed to capture both its name attribute and wsdl:documentation sub-element. This way, the Service property provides information related to the operation's service as a whole. Similarly, the wsdl:operation element is mapped to the Capability property (see line 13), which describes the functionality offered by the operation.

1. http://www.w3.org/TR/wsdl

Algorithm A.1: Translate a given WSDL document that is accessible by a URL into one or more Proteus service advertisements.

```
input: the URL of a WSDL document, $url$
result: a set of Proteus service advertisements, $\mathcal{S}_A$
1  begin
2      $D \leftarrow$ get WSDL document from $url$
3      $d \leftarrow$ get definitions element from $D$
4      $S \leftarrow$ get all service elements from $d$
5      foreach $s \in S$ do
6          $P \leftarrow$ get all port elements from $s$
7          foreach $p \in P$ do
8              $O \leftarrow$ get all operation elements from $p$
9              foreach $o \in O$ do
10                 $A \leftarrow$ new service advertisement
11                 Service $\leftarrow$ map ($s$)
12                 add Service to $A$
13                 Capability $\leftarrow$ map ($o$)
14                 add Capability to $A$
15                 if $o$ has input message then
16                     $m_i \leftarrow$ get input message from $o$
17                     Input $\leftarrow$ map ($m_i$)
18                     add Input to $A$
19                 end
20                 if $o$ has output message then
21                     $m_o \leftarrow$ get output message from $o$
22                     Output $\leftarrow$ map ($m_o$)
23                     add Output to $A$
24                 end
25                 WSInvocationInfo $\leftarrow$ map ($s$, $p$, $o$, $url$)
26                 add WSInvocationInfo to $A$
27                 add $A$ to $\mathcal{S}_A$
28             end
29         end
30     end
31 end
```

The wsdl:message element that corresponds to the operation's input message is mapped to the Input property (see line 17). In addition to the use of the *string value* and *semantics* features in the same way as previously described, this property also uses the *sub-properties group* feature to capture the wsdl:part sub-elements of the wsdl:message element. The content of each such sub-element is mapped to a Parameter property, as follows:

- The name attribute is mapped to both the *string value* and *semantics* features of the Parameter.
- The type, or element, attribute is mapped to the *data type* feature of the Parameter.

The mapping of the wsdl:message element that corresponds to the operation's output message to the Output property is performed in a similar manner (see line 22). In other words, the resulting Input and Output properties have the same structure.

Finally, the WSInvocationInfo property employs the *qualifiers* feature, which is populated with the ServiceName, PortName, OperationName, and WSDL properties capturing the name attribute values of the wsdl:service, wsdl:port, and wsdl:operation elements, as well as the URL of the WSDL document, respectively (see line 25). Specifically, each one of these attribute values is assigned to the *string value* feature of its corresponding property.

In conclusion, our WSDL translation algorithm and mapping rules manage to preserve all meaningful information in the generated Proteus service advertisements. This information is captured by the *Service*, *Capability*, *Input*, *Output*, and *WSInvocationInfo* properties, and serves the purposes of both service discovery and service invocation.

A.2 Translating WSRF-enhanced WSDL Documents

The WSDL standard was also adopted by the Grid community in order to describe the Grid services that are offered by virtual organizations. The Web Services Resource Framework (WSRF)[2], which has become a standard for the implementation of service-oriented Grids, supports the stateful interactions with service consumers, by establishing a strong relation between a provided functionality and its underlying *resource*. The latter has one or more *resource properties*, which are included in the WSDL description by extending the portType element(s).

Hence, Algorithm A.1 can be easily adapted to map WSRF-enhanced WSDL documents into their equivalent Proteus service advertisements. Specifically, right after line 6, we introduce an additional step to retrieve the value of the wsrp:ResourceProperties attribute of the portType that is associated with the currently processing port, p. This value points at a complex typed XSD element in the types section of the WSDL document, which is mapped to the *Resource* property in the Proteus service advertisement, as follows:

- The name attribute of the XSD element is mapped to the *string value* and *semantics* features of the Resource property.
- Each one of the XSD sub-elements in the element's complex type are captured by a ResourceProperty property, by mapping their name attribute to the property's *string value* and *semantics* features, while their type attribute is mapped to the property's *data type* feature. Also, the minOccurs and maxOccurs attributes are mapped to the *numeric value* features of their namesake properties, while the latter are grouped by the *qualifiers* feature of the ResourceProperty property. Finally, all produced ResourceProperty properties are consolidated into the *subproperties group* feature of the Resource property.

2. http://www.oasis-open.org/committees/wsrf/

A.3 Translating OWL-S Documents

The OWL-S upper ontology for services[3] has been established over the last years as a fine-grained solution for the semantic description of Web services. Briefly, the ontology defines three top-level OWL classes, namely the *ServiceProfile*, the *ServiceModel*, and the *ServiceGrounding*, which are used to semantically describe a Web service from different perspectives, and accommodate different needs. Since only the *ServiceProfile* class is intended to facilitate service discovery, we describe in this section the translation of an OWL-S 1.1 Profile document to its equivalent Proteus service advertisement.

Algorithm A.2: Translate a given OWL-S Profile document into its equivalent Proteus service advertisement. It is assumed that the OWL-S Profile document contains only one Profile element.

```
input: an OWL-S Profile document, $D$
result: a Proteus service advertisement, $A$
1  begin
2      $A \leftarrow$ new service advertisement
3      $P_s \leftarrow$ get Profile element from $D$
4      $s \leftarrow$ get presentedBy element from $P_s$
5      $p_s \leftarrow$ get has_process element from $P_s$
6      $c_s \leftarrow$ get serviceClassification element from $P_s$
7      $\pi_s \leftarrow$ get serviceProduct element from $P_s$
8      $\kappa_s \leftarrow$ get serviceCategory element from $P_s$
9      $C_s \leftarrow$ get contactInformation elements from $P_s$
10     Service $\leftarrow$ map ($s$, $p_s$, $c_s$, $\pi_s$, $\kappa_s$, $C_s$)
11     add Service to $A$
12     $n_s \leftarrow$ get serviceName element from $P_s$
13     $d_s \leftarrow$ get textDescription element from $P_s$
14     Capability $\leftarrow$ map ($n_s$, $d_s$)
15     add Capability to $A$
16     $I \leftarrow$ get hasInput elements from $P_s$
17     Input $\leftarrow$ map ($I$)
18     add Input to $A$
19     $O \leftarrow$ get hasOutput elements from $P_s$
20     Output $\leftarrow$ map ($O$)
21     add Output to $A$
22     $P \leftarrow$ get hasPrecondition elements from $P_s$
23     Preconditions $\leftarrow$ map ($P$)
24     add Preconditions to $A$
25     $E \leftarrow$ get hasResult elements from $P_s$
26     Results $\leftarrow$ map ($E$)
27     add Results to $A$
28     $S_p \leftarrow$ get all serviceParameter elements from $P_s$
29     foreach $s_p \in S_p$ do
30         $p \leftarrow$ new Proteus property
31         $n \leftarrow$ get serviceParameterName from $s_p$
32         $o \leftarrow$ get sParameter from $s_p$
33         set $n$ as the name of $p$
34         set $o$ as the semantics of $p$
35         add $p$ to $A$
36     end
37 end
```

As it can be seen in our Algorithm A.2, the translation of an OWL-S profile maps its contents to the following properties in the generated Proteus service advertisement: *Service*, *Capability*, *Input*, *Output*, *Preconditions*, and

3. http://www.w3.org/Submission/OWL-S/

Results. Further, as we will see, extensibility elements of the OWL-S profile are also preserved thanks to the generic nature of the Proteus service property structure.

The Service property is conveniently used as a container for the mapping results of numerous elements found in the OWL-S profile (see lines 4–10). Specifically, the presentedBy element, which points at the service implementation of the profile, is mapped to the *semantics* feature of the property. Then, the *qualifiers* feature is employed to host the mapping results for the has_process, serviceClassification, serviceProduct, serviceCategory, and contactInformation elements. For each one of these elements, a corresponding property is generated and added to the qualifiers of the Service, as follows:

- The value of the has_process element, which is the URI of the corresponding concept in the Process ontology, is mapped to the *semantics* feature of the Process property.
- The value of the serviceClassification element is mapped to the *semantics* feature of the Classification property.
- The value of the serviceProduct element is mapped to the *semantics* feature of the Product property.
- The values of the categoryName, taxonomy, and code sub-elements of the serviceCategory element are mapped to the *string value* features of their equivalent properties, Name, Taxonomy, and Code, which are then grouped by the *qualifiers* feature of the Category property, while the value sub-element is mapped to the property's *string value* feature.
- The Actor sub-element of each contactInformation element is mapped to a Contact property. The values of the name, title, phone, fax, email, physicalAddress, and webURL sub-elements of the Actor are mapped to the *string value* features of their equivalent properties, Name, Title, Phone, Email, PhysicalAddress, and WebURL. The latter are then grouped by the Contact property through the *qualifiers* feature. Finally, the generated Contact properties are grouped by the ContactInformation property through the *sub-properties group* feature. This way, we manage to map all contactInformation elements into a single property in the Proteus service advertisement.

The Capability property captures the information conveyed by the serviceName and textDescription elements of the OWL-S profile (see lines 12–14). Specifically, the serviceName value is mapped to both the *string value* and *semantics* features of the Capability, whereas the textDescription value is mapped to the *semantics* feature only. In other words, the value of the *semantics* feature of the Capability property is a concatenation of the values of the serviceName and textDescription elements.

All the hasInput elements found in the OWL-S profile are mapped to Parameter properties, which are subsequently grouped into a single Input property, with the use of the *sub-properties group* feature. Each Parameter makes use of the *semantics* feature to capture the value of the resource attribute of its corresponding hasInput element. In the same way, the hasOutput elements are mapped to a single Output property in the Proteus service advertisement. Also following similar mapping rules, the hasPrecondition and hasResult elements are mapped and accordingly grouped to the Preconditions and Results properties, respectively.

Finally, the expandable list of serviceParameter elements that may be specified in an OWL-S profile are naturally mapped to Proteus properties, as described in lines 28–36 of Algorithm A.2. To summarize, our algorithm and mapping rules presented in this section manage to translate an OWL-S 1.1 profile document into its equivalent Proteus service advertisement, preserving all included information.

A.4 Translating hRESTS Microformats

The HTML for RESTful Services (abbreviated to hRESTS)[4] is a microformat that is used for the machine-readable description of Web APIs. hRESTS is underpinned by a minimal service model, which defines a small set of concepts along with their relations. According to this model, a *service* has one or more *operations*, while each operation consumes zero or more *inputs*, and produces zero or more *outputs*. Further, each operation has a URI *address*, and an HTTP *method*, which are required by clients in order to invoke it.

The aforementioned concepts are captured by a set of XHTML classes, which give shape to the basic hRESTS microformat. Nevertheless, it is possible to blend an hRESTS description with other microformats so as to further annotate a service with additional information, such as semantics or QoS. For instance, the MicroWSMO microformat can be used to semantically annotate a RESTful service that is described with hRESTS. Given the RDF form of an XHTML document with hRESTS and MicroWSMO microformat markups, Algorithm A.3 is applied to generate one or more Proteus service advertisements. The algorithm produces one Proteus service advertisement for each hasOperation property found in the given RDF document (see lines 4–26). Each service advertisement is populated with the properties *Service*, *Capability*, *RESTInvocationInfo*, *Input*, and *Output*.

The Service property captures the information conveyed by its namesake RDF resource (lines 7–8). Specifically, the *string value* feature is employed to capture the local name of the RDF Service resource URI, while the *semantics* feature is used to capture (i) the local name of the RDF Service resource URI concatenated with the value of the RDFS label sub-property, and (ii) the value of the MicroWSMO modelReference sub-property. In exactly the same way, the RDF Operation resource is

4. J. Kopecký, K. Gomadam, and T. Vitvar, hRESTS: An HTML Microformat for Describing RESTful Web Services. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, pp. 619–625, IEEE Computer Society, 2008.

Algorithm A.3: Translate a given RDF representation of a RESTful service description in hREST and MicroWSMO, into its equivalent Proteus service advertisements.

```
input: the RDF document of hRESTS description, $D$
result: a set of Proteus service advertisements, $S_A$
begin
    $s \leftarrow$ get Service resource from $D$
    $P_O \leftarrow$ get all hasOperation properties of $s$
    foreach $p \in P_O$ do
        $o \leftarrow$ get Operation resource pointed by $p$
        $A \leftarrow$ new service advertisement
        Service $\leftarrow$ map ($s$)
        add Service to $A$
        Capability $\leftarrow$ map ($o$)
        add Capability to $A$
        $a \leftarrow$ get hasAddress property of $o$
        $m \leftarrow$ get hasMethod property of $o$
        RESTInvocationInfo $\leftarrow$ map ($a$, $m$)
        add RESTInvocationInfo to $A$
        $I \leftarrow$ get all hasInputMessage properties of $o$
        if $I \neq \emptyset$ then
            Input $\leftarrow$ map ($I$)
            add Input to $A$
        end
        $O \leftarrow$ get all hasOutputMessage properties of $o$
        if $O \neq \emptyset$ then
            Output $\leftarrow$ map ($O$)
            add Output to $A$
        end
        add $A$ to $S_A$
    end
end
```

mapped to the Capability property in the Proteus service advertisement (lines 9–10).

In order to invoke an operation offered by a RESTful service, clients primarily need to know its corresponding address and HTTP method. The RESTInvocationInfo property is introduced to sustain that information in the produced Proteus service advertisement (lines 11–13). The algorithm maps the RDF resources pointed by the hasAddress and hasMethod properties of the RDF Operation resource, into the *string value* features of the corresponding properties, Address and HTTPMethod. The latter are then added to the *qualifiers* feature of the RESTInvocationInfo property.

In hRESTS, each input or output parameter of an operation corresponds to an RDF Message resource. Hence, to capture the inputs/outputs of the service operation, the algorithm generates an Input/Output property, and maps all input/output messages to their corresponding Proteus Parameter properties. The latter are consolidated by the *sub-properties group* feature of the Input/Output property (see lines 15–24). The contents of each Parameter property are shaped as follows:

- The local name of the corresponding RDF Message resource is mapped to the *string value* feature.
- Both the local name and the RDFS label property of the RDF Message are mapped to the text part of the *semantics* feature.
- The modelReference property of the RDF Message is mapped to the ontology reference part of the *semantics* feature.
- The lowerSchemaMapping and liftingSchemaMapping properties of the RDF Message are mapped to the *string value* features of their namesake Proteus properties, while the latter are grouped by the *qualifiers* feature of the Parameter property.

In conclusion, the presented algorithm and mapping rules capture the contents of an hRESTS description given in RDF format, thus producing one or more Proteus service advertisements that correspond to the declared operations of the RESTful service, without any loss of information.

A.5 Translating JXTA MSA Documents

JXTA[5] is a collection of programming language– and platform–independent protocols for P2P computing. According to these protocols, peers expose their functionality through P2P services which they appropriately publish within the network, so that other peers can discover and invoke them. To support the service publication and discovery processes, the JXTA specification defines the *Module Spec Advertisement (MSA)*, a simple, XML-based document that is used for the description of P2P services. Although MSA documents typically contain minimal information about a service, they are the default choice for service description in JXTA, and are supported by the standard service discovery and invocation mechanisms of the JXTA specification. Thus, for the Proteus query model to become applicable to P2P service discovery in JXTA environments, there is a need of properly translating MSA documents into Proteus service advertisements.

Due to the simplicity of MSA documents, the overall translation process is straightforward, and is captured by Algorithm A.4. In addition to the MSA document, the translator also expects the peer and peer group identifiers as input, so as to associate the MSA with its owner peer and the group it belongs to. Such data are easily obtained by the Proteus crawler ahead of the translation, by issuing appropriate queries via the default JXTA discovery mechanism, since JXTA advertisements are automatically indexed on the basis of their peer and peer group ids. Nevertheless, as it can be seen in the algorithm, the information contained in the MSA document is mapped and organized into four properties in the produced Proteus service advertisement, namely the *Peer*, *Service*, *Capability*, and *JXTAInvocationInfo*.

We introduce the Peer property to capture the id of the peer owning the MSA, as well as the id of the peer's belonging group. These values are mapped to the *string value* features of the PeerID and PeerGroupID properties, respectively, which are then grouped by the *qualifiers* feature of the Peer property.

5. http://jxta.kenai.com/

Algorithm A.4: Translate a given JXTA *Module Spec* Advertisement (MSA) into its equivalent Proteus service advertisement. The *id*s of the peer owning the corresponding service, as well as its peer group are also provided as input.

```
input: a JXTA MSA document, $D$
input: the id of the owner peer, $pid$
input: the id of the peer group, $gid$
result: a Proteus service advertisement, $A$
begin
    $A \leftarrow$ new service advertisement
    $id_A \leftarrow$ get MSID element from $D$
    Peer $\leftarrow$ map ($pid$, $gid$)
    add Peer to $A$
    $vers \leftarrow$ get Vers element from $D$
    $crtr \leftarrow$ get Crtr element from $D$
    $suri \leftarrow$ get SURI element from $D$
    Service $\leftarrow$ map ($vers$, $crtr$, $suri$)
    add Service to $A$
    $name \leftarrow$ get Name element from $D$
    $desc \leftarrow$ get Desc element from $D$
    if $name \neq$ null or $desc \neq$ null then
        Capability $\leftarrow$ map ($name$, $desc$)
        add Capability to $A$
    end
    $pipe \leftarrow$ get PipeAdvertisement element from $D$
    $proxy \leftarrow$ get Proxy element from $D$
    $auth \leftarrow$ get Auth element from $D$
    if $pipe \neq$ null or $proxy \neq$ null or $auth \neq$ null then
        JXTAInvocationInfo $\leftarrow$ map ($pipe$, $proxy$, $auth$)
        add JXTAInvocationInfo to $A$
    end
end
```

The Service property is used to map a number of elements of the MSA that provide general information about the P2P service (see lines 6–9). Specifically, the Vers, Crtr, and SURI elements are mapped to the *string value* features of the respective properties Version, Creator, and SpecURI, which are followingly grouped by the *qualifiers* feature of the Service property.

The Name and Desc elements in the MSA are mapped to the Capability property in the Proteus service advertisement (see lines 11–16). The value of the Name is mapped to both the *string value* and *semantics* features, while the value of the Desc is mapped to the *semantics* feature only. Besides, since the MSA may contain information that is useful to the invocation of the corresponding P2P service, we employ the JXTAInvocationInfo property to capture the respective elements (see lines 17–23), as follows:

- The PipeAdvertisement element is mapped to a property with the same name. Specifically, the values of the Id, Type, Name, and Desc sub-elements are mapped to the *string value* features of their equivalent, namesake properties, which are grouped into the *qualifiers* feature of the PipeAdvertisement property.
- The values of the Proxy and Auth elements are mapped to the *string value* of the ProxySpecID and AuthenticatorSpecID propeprties, respectively.

Finally, the PipeAdvertisement, ProxySpecID, and AuthenticatorSpecID properties are grouped into the *qualifiers* feature of the JXTAInvocationInfo property.

A.6 Capturing QoS descriptions

This section demonstrates the ability of Proteus to capture non-functional service properties, in addition to functional properties such as the capability, inputs and outputs of a service operation. As example, we have selected the WS-QoS[6] schema, which provides a series of elements allowing for the annotation of WSDL documents, and potentially other kinds of service description formats, with QoS information.

The WS-QoS specification supports the expression of as many as 10 application-level QoS properties, such as the service availability, reliability, average processing time, etc. All these elements can be fully captured by the following properties within a Proteus service advertisement:

- The value of the price element, included in the QoSDefinition element of a WS-QoS document, is mapped to the *numeric value* feature of the Price property. Moreover, the currency attribute of the price element is captured by the *string value* feature of the Currency property, which is included in the *qualifiers* feature of the Price property.
- The values of the processingTime, requestsPerSecond, availability, and reliability elements, both included in the serverQoSMetrics container of a WS-QoS document, are mapped to the *numeric value* features of their namesake properties.
- The values of the delay, jitter, throughput, packetLoss are mapped to the *numeric value* features of namesake properties in the Proteus service advertisement. Furthermore, the values of their min and max sub-elements are mapped to the *numeric value* features of their namesake sub-properties, which are then included in the *qualifiers* feature of the respective properties.

Besides, the extensibility of the WS-QoS schema, which is particularly supported by the customMetric and customPriority elements, is retained during the translation process. Instances of these elements are preserved in the Proteus service advertisement as follows: Their name attribute is used as the name of the corresponding property, whereas their ontology attribute is mapped to the *semantics* feature of that property. This way, we manage to fully incorporate the contents of a WS-QoS document into a Proteus service advertisement, without any loss of information.

6. M. Tian, A. Gramm, H. Ritter, and J. Schiller, Efficient Selection and Monitoring of QoS-aware Web Services with the WS-QoS Framework, In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence, WI '04*, pp. 152–158, IEEE Computer Society, 2004.