WSDL-S: A Proposal to W3C WSDL 2.0 Committee

John Miller, Kunal Verma, Preeda Rajasekaran, Amit Sheth, Rohit Aggarwal, Kaarthik Sivashanmugam
LSDIS Lab, University of Georgia

1. **Introduction:** the need for adding semantics to WSDL

Web services have primarily been designed for providing inter-operability between business applications. Current technologies assume a large amount of human interaction, for integrating two applications. This is primarily due to the fact that business process integration requires understanding of data and functions of the involved entities. Semantic Web technologies, powered by description logic based languages like OWL[1], aim to add greater meaning to Web content, by annotating the data with ontologies. Ontologies provide a mechanism of providing shared conceptualizations of domains. This allows agents to get an understanding of users' Web content and greatly reduces human interaction for meaningful Web searches. A similar approach can be used for adding greater meaning to Web service descriptions, which will in turn, allow greater automation, by reducing human involvement for understanding the data and functions of the services,

Several standards have been proposed for creating semantic Web services. Primarily, OWL-S[2][8], WSMO[3] and METEOR-S [4] have proposed various solutions for creating for expressive descriptions for Web services. In this draft, we present WSDL-S, a lightweight approach for adding semantics to Web services. This draft is based on the ongoing METEOR-S project, LSDIS Lab, University of Georgia, which aims to add semantics to the complete lifecycle of Web processes.

Section 2 outlines WSDL and how semantics can further enhance it. Section 3 discusses on adding semantics to web services. Section 4 describes the WSDL-S Meta-Model while Section 5 explains a step-by-step approach of adding semantics. Appendices provide further details and corroborating information. Appendix I shows an example WSDL-S file while an example of corresponding annotated source code is provided in Appendix II. Appendix III details the interim step between WSDL and WSDL-S. Appendix IV lists the tools that are being built under METEOR-S to support WSDL-S while Appendix V shows the WSDL-S support in METEOR-S.

2. **WSDL**

The WSDL specification began with WSDL 1.0 in 2000. This was quickly followed by WSDL 1.1[5] which has been used in many Web service tools. A effort to enhance it, WSDL 1.2, has been renamed WSDL 2.0[6], as it includes some major changes. Draft releases of the WSDL 2.0 standard have been available since March of 2004 and are expected to become W3C recommendation in 2005.

A central purpose of WSDL is to describe interfaces (formerly known as port-types) to Web services. Actual Web services may then be viewed as remote implementations of these of interfaces. In general, service providers/implementers could use a standard interface, extend a standard interface or develop there own.

Broadly speaking, an interface contains a set of operations. Each operation has a signature which includes an operation name, input, output and exception messages. These messages have types that are defined using some XML-based schema language. The schema language that is commonly used is XSD[7], although OWL is an alternative. In WSDL 2.0, types are pushed more completely outside the standard. This makes sense on the one

hand, since types systems are complex to define and there exist at least two well-accepted type systems in the XML world: XSD and OWL. On the other, this makes it difficult for WSDL interfaces to be first-class citizens (i.e., types) which they would be under an object-oriented approach.

A client of a Web service will look to the interface to find out what it will do. Indeed, interface descriptions may be used to find candidate Web services. Such descriptions are therefore critical to proper discovery and use of Web services. Hence, adding semantics to interfaces is very important. To illustrate this idea, more semantics is added to Web service interface in a step by step fashion. Note, the examples are based on Java source for brevity.

1. Minimal semantics.

```
interface Interface1WS
{
    String operation1 (String param);
    String operation2 (String param);
} // Interface1WS
```

2. Use meaningful argument names and more specific types.

```
interface BookWS
{
    double getPrice (String isbn, String title, int year)
    boolean buy (String ISBN);
} // BookWS
```

3. Require types to be from (or extend) standard libraries. Upcast to existing types is automatic, while one must supply a method for downcast. For language independence, these types need to be standardized for an XML based schema language, such as XSD or OWL. Conversion from one type system (XSD, OWL and Java) to another is currently an open problem.

```
import BookTypes.*;
interface BookWS
{
    double getPrice (BookID isbn, BookTitle title, Date year)
    boolean buy (BookID ISBN);
} //BookWS
```

4. Add Design-By-Contract (nominally pre and post conditions)

```
import BookTypes.*;
interface BookWS
{
    @post (return > 0.0);
```

```
  double getPrice (BookID isbn, BookTitle title, Date year);
  @pre (isbn != null);
  boolean buy (BookID isbn);
} //BookWS
```

3. **Adding Semantics to Web services**

The OWL-S (formerly DAML-S) project defines an ontology for the domain of Web services. This ontology provides tags which can be used for describing actual Web services. The ontology consists of three sub ontologies, service profile, service grounding and the process model, which are tied together using a service ontology. The service profile defines the functional and non-functional properties of the services. Service grounding contains information about invocation. In an effort to align with industry standards, service grounding provides mapping OWL atomic processes to WSDL operations. The process model describes the ordering of the operations of the service.

Due to the late alignment of OWL-S to WSDL, the design of OWL-S may be unnecessary complex. WSDL-S aims to provide a lightweight approach for creating semantic Web service descriptions. We call our approach lightweight because:
- We provide simple extensions to WSDL to add semantics, thereby allowing semantic descriptions of Web services.
- We have designed the WSDL-S service ontology, which is more aligned with WSDL 2.0 and more compact than OWL-S, without losing significant expressivity. In particular, we were not convinced on separating the service grounding from the service profile. An important feature of our approach is that we automatically populate our ontology from WSDL 2.0 descriptions. We view our ontology as an intermediate entity and plan to shield the users from interacting with it.
- Our approach allows integration of semantic and non semantic descriptions of Web services, as users using special types must specify translation to OWL

4. **WSDL-S Meta – Model**

In this section, we present a meta-model for WSDL 2.0. Our extensions have been shown in red and green. In near future, we will present further refinements to this meta–model and an approach to represent the process model.

The tags in red are our proposed extensions. We explain them in detail in the next section. WSDL 2.0 draft already discusses using different type systems for Web services. We have shown OWL and XMI as possible type systems, along with XSD in the meta-model. Since WSDL 2.0 partially supports this, we have shown them in green. We are still investigating efficient ways for transforming from one type system to the other.
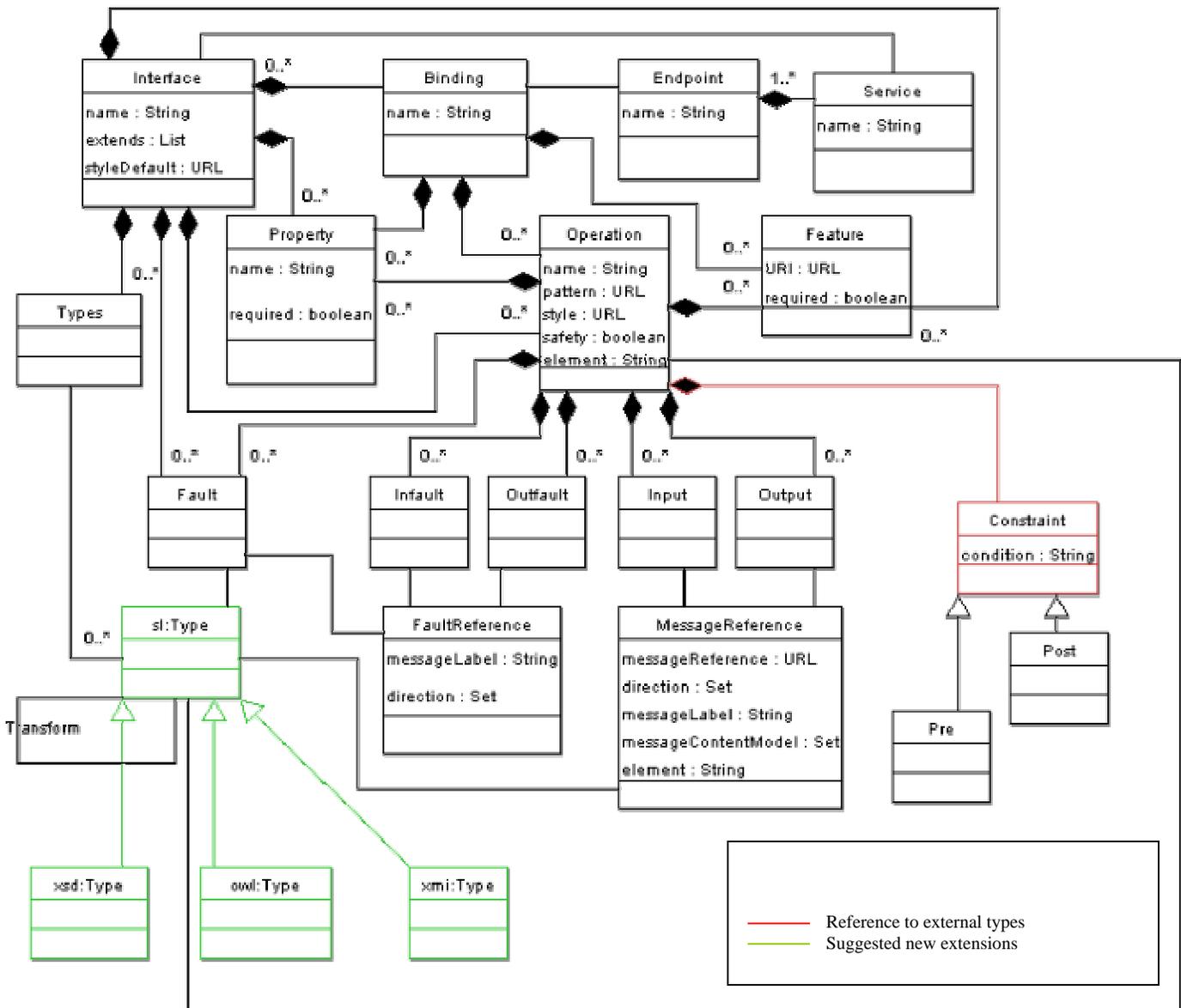
Figure 1: Meta-model for WSDL 2.0 with proposed extensions

5. **Adding Semantics: Step – by – step**

In this section, we will explain our approach of adding semantics to WSDL 2.0.

5.1. **Referencing Ontologies in WSDL-S**

We present examples using an ontology based on the Rosetta net PIP directory. An initial draft on the ontology is available at http://webster.cs.uga.edu/~azami/pips.owl. We use the namespace feature of WSDL to reference classes and properties from this ontology. Below, we show an example of creating the namespace "rosetta" for the Rosetta Net ontology.

```
<definitions
        .
        .
        xmlns:rosetta = http://webster.cs.uga.edu/~azami/pips.owl#   ……      />
```

5.2. **Adding semantics to operations**

In WSDL 2.0, each interface consists of a number of operations. Each operation can be seen as a unit of functionality of the service. It is imperative to capture the functionality of the each operation. In order to illustrate our extensions, let us consider an operation which allows users to cancel an order. WSDL 2.0 fragment for this particular operation is shown below.

```
<operation name = "checkStatus" pattern="mep:in-out" >
    <action element = "rosetta:QueryOrderStatus"  />
    <input messageLabel = "statusQuery" element  = "xsd:PurchaseOrderStatusQuery" />
    <output messageLabel = "status" element  = "xsd:PurchaseOrderStatusResponse" />
 </operation>
```

In this section, we show the corresponding WSDL-S representation for the above fragment. A complete WSDL-S interface is shown in Appendix I. We have used extensibility of WSDL 2.0 to add two child (shown in red) tags to the "operation" tag. They are:

1. **"Action"** tag depicts the action the operation performs. This operation which allows a client to cancel an order, is depicted using PIP 3A9 Request Purchase Order Cancellation. We use the action tag to point to the corresponding class in the Rosetta Net ontology.

2. **"Constraint"** tag is used to depict pre and post conditions. In this case the confirmation number must be greater than zero for the operation to be executed. So it is depicted using the pre tag.

```
<operation name = "checkStatus" pattern="mep:in-out" >
    <action element = "rosetta:QueryOrderStatus"  />
    <input messageLabel = "statusQuery" element  = "rosetta:PurchaseOrderStatusQuery" />
    <output messageLabel = "status" element  = "rosetta:PurchaseOrderStatusResponse" />
   <pre condition="PurchaseOrderStatusQuery.orderStatusDoc.?PurchaseOrder !=null" />
 </operation>
```

In addition, we depict the inputs and outputs using OWL types (shown in bold) from the Rosetta Net ontology instead on XML schema types (XSD).

Summary

In our current implementation, we can
1. Create WSDL-S from annotated source code
2. Automatically publish WSDL-S in a enhanced UDDI registry.
3. Use WSDL-S files to automatically generate OWL-S files for grounding, profile and service.

We expect to add the following capabilities in the near future
1. Generate automated Java source files (either interfaces or abstract classes) from WSDL-S
2. Generate a client to invoke a Web service using WSIF.
3. We are also working on adding semantics to WSDL-S or a related document in order to generate an OWL-S process for the service (note, this is related to general area of protocol specification).

Our overall goal is to use WSDL-S in Web service orchestration where the individual Web services are used in larger Web processes. We are particularly interested in automating aspects of Web process creation. Precise and adequate semantics are essential for automation. Our research in this matter will provide feedback to the adequacy of WSDL-S.

References:

[1]OWL Web Ontology Language Overview - http://www.w3.org/TR/2003/CR-owl-features-20030818/
[2] OWL-S 1.0 Release - http://www.daml.org/services/owl-s/1.0/
[3] WSMO-Web Service Modeling Ontology - http://www.wsmo.org
[4] METOR-S : Semantic Web Services and Processes - http://lsdis.cs.uga.edu/Projects/METEOR-S/
[5]WSDL1.1 - Web Services Description Language (WSDL) 1.1 - http://www.w3.org/TR/wsdl
[6]WSDL2.0 - Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language-
    http://www.w3.org/TR/2004/WD-wsdl20-20040326/
[7]XSD – XML - Schema http://www.w3.org/TR/2004/WD-wsdl20-20040326/
[8] Sirin, Evren and and Hendler, James and Parsia, Bijan. Semi-automatic Composition of Web Services using Semantic Descriptions.   2003.

APPENDIX I - ANNOTATED WSDL file (WSDL-S)

The extensible tags to hold the semantic information are entered according to the WSDL 2.0 standards grammar. The semantic and non-semantic information in the WSDL file is obtained from the annotated source code (see appendix II).

```xml
<?xml version="1.0" encoding="UTF-8"?>

<definitions
      name = "BatterySupplier"
      targetNamespace = "http://lsdis.cs.uga.edu/meteor/BatterySupplier.wsdl20"
      xmlns = "http://www.w3.org/2004/03/wsdl"
      xmlns:tns = "http://lsdis.cs.uga.edu/BatterySupplier.wsdl20"
      xmlns:rosetta = "http://webster.cs.uga.edu/~azami/pips.owl#"
      xmlns:mep=http://www.w3.org/TR/wsdl20-patterns>

   <interface name = "BatterySupplierInterface" >

     <operation name = "getQuote"  pattern = "mep:in-out"  >
         <action  element = "rosetta:RequestQuote" />
        <input messageLabel = "qRequest" element = "rosetta:QuoteRequest" />
        <output messageLabel = "quote" element  = "rosetta:QuoteConfirmation"  />
      </operation>

      <operation name = "placeOrder"  pattern = "mep:in-out"  >
         <action element = "rosetta:RequestPurchaseOrder" />
        <input messageLabel = "order" element = "rosetta:PurchaseOrderRequest" />
        <output messageLabel = "orderConfirmation" element  = "rosetta:PurchaseOrderConfirmation" />
      </operation>

      <operation name = "checkStatus" pattern="mep:in-out" >
         < action  element = "rosetta:QueryOrderStatus" />
        <input messageLabel = "statusQuery" element  = "rosetta:PurchaseOrderStatusQuery" />
        <output messageLabel = "status" element  = "rosetta:PurchaseOrderStatusResponse" />
        <pre condition="statusQuery.orderStatusDoc.?PurchaseOrder !=null" />
      </operation>

   </interface>
</definitions>
```

APPENDIX II – Annotated Java Source Code

For software developers, it is convenient to allow them to annotate source files to enable complete WSDL-S specifications to be generated. Incorporation of semantics in the source code is achieved by exploiting the meta-tag feature of the latest release of j2se, j2se1.5. Source code may be used as the central place-holder for Web services development, deployment and publication. Given below is the annotated source code corresponding to the WSDL-S specifications given in Appendix I.

```java
import java.lang.annotation.*;
import java.lang.reflect.*;

@namespaces ({
@namespace( name = "rosetta", service_extension = true, url = "http://webster.cs.uga.edu/~azami/pips" )
 })

public interface BatterySupplier {

   @operation ( name = "getQuote", action = "rosetta:RequestQuote" )
   @parameters ({
      @inParam ( name = "qRequest", element = "rosetta:QuoteRequest" ),
      @outParam ( name = "quote",  element =  "rosetta:QuoteConfirmation" )
    })

   QuoteConfirmation requestQuote (QuoteRequest qRequest );

   @operation  ( name = "placeOrder", action  = "rosetta:RequestPurchaseOrder" )
   @parameters ({
      @inParam ( name = "order", element = "rosetta:PurchaseOrderRequest" ),
      @outParam ( name = "orderconfirmation", element = "rosetta:PurchaseOrderConfirmation"  )
    })

   PurchaseOrderConfirmation placeOrder(PurchaseOrderRequest order );

   @operation ( name = "checkStatus", action = "rosetta:QueryOrderStatus " )
   @parameters ({
      @inParam ( name = "statusQuery",  element =" rosetta:PurchaseOrderStatusQuery"  ),
      @outParam  ( name = "status",  element = "rosetta:PurchaseOrderStatusResponse" )
   })
      @pre ( condition = "statusQuery.orderStatusDoc.?PurchaseOrder !=null" )

   PurchaseOrderStatusResponse checkStatus (PurchaseOrderStatusQuery statusQuery);

 }//end of class
```

APPENDIX III - INTERIM STEP (annotated WSDL 1.1)

As an interim step of defining WSDL-S, we have added semantics to WSDL 1.1 by using the extensibility elements offered by the WSDL 1.1 standard. The inputs and output are annotated by adding a reference to ontology concepts in the <part> tag under the <message> tag of WSDL 1.1. For the operation concept, the operation tag is extended to add a reference to the functional ontology. For other service details like description, geographic location and category classification, the LSDISExt:serviceDetails tag is used. Given below is an annotated WSDL1.1 file corresponding to the WSDL-S specification given in Appendix I.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
       name = "BatterySupplier"
       targetNamespace = "http://lsdis.cs.uga.edu/meteor/BatterySupplier.wsdl20"
       xmlns = http://www.w3.org/2004/03/wsdl
       xmlns: lsdisxd="http://lsdis.cs.uga.edu/complexTypes"
       xmlns:tns = "http://lsdis.cs.uga.edu/BatterySupplier.wsdl20"
       xmlns:rosetta = "http://webster.cs.uga.edu/~azami/pips.owl#"
       xmlns:LSDISOnt="http://banking.ontology.com/onto_concepts"
       xmlns:LSDISExt="http://wwww.cs.uga.edu/~aggarwal/service_ontology"
       xmlns:mep="http://www.w3.org/TR/wsdl20-patterns">

       <message name = "qRequest">
              <part name="in0" type="lsdisxd:QuoteRequest"
                LSDISExtnto-concept = "rosetta:QuoteRequest">
       </message>
       <message name = " quote">
              <part name="in0" type="lsdisxd:QuoteConfirmation"
                LSDISExtnto-concept = " rosetta:QuoteConfirmation ">
       </message>
       <message name = " order">
              <part name="in0" type="lsdisxd:PurchaseOrderRequest"
                LSDISExtnto-concept = " rosetta:PurchaseOrderRequest ">
       </message>
       <message name = " orderConfirmation">
              <part name="in0" type="lsdisxd:PurchaseOrderConfirmation"
                LSDISExtnto-concept = " rosetta:PurchaseOrderConfirmation ">
       </message>
       <message name = " statusQuery">
              <part name="in0" type="lsdisxd:PurchaseOrderStatusQuery"
                LSDISExtnto-concept = "rosetta: PurchaseOrderStatusQuery ">
       </message>
       <message name = " status">
              <part name="in0" type="lsdisxd:PurchaseOrderStatusResponse"
                LSDISExtnto-concept = "rosetta: PurchaseOrderStatusResponse ">
       </message>
```

```
<interface name = "BatterySupplierInterface" >
     <operation name="getQuote" LSDISExt:operation-expose="true"
                                   LSDISExt:onto-concept="rosetta:RequestQuote">
          <input messageLabel ="qRequest" />
          <output messageLabel ="quote" />
     </operation>

     <operation name=" placeOrder " LSDISExt:operation-expose="true"
                                   LSDISExt:onto-concept=" rosetta:RequestPurchaseOrder ">
          <input messageLabel ="order" />
          <output messageLabel ="orderConfirmation" />
     </operation>

     <operation name="checkStatus" LSDISExt:operation-expose="true"
                                   LSDISExt:onto-concept=" rosetta:QueryOrderStatus ">
          <input messageLabel = "statusQuery" />
          <output messageLabel = "status" />
          <pre condition="statusQuery.orderStatusDoc.?PurchaseOrder !=null" />
     </operation>

   </interface>
</definitions>
```

APPENDIX IV - TOOLS

The following standalone tools are being built under the METEOR-S project, which would help produce WSDL-S documents manually and semi-automatically.

1. WSDL to WSDL-S converter: This would be an eclipse plug-in which would take a WSDL file and the users can manually suggest ontology concepts for each input, output and operation in the WSDL. The toll will then generate a WSDL-S file. Estimate release date: July 31, 2004.

2. Annotated Source code to WSDL-S converter: This tool will take an annotated source code file as input and will automatically produce a WSDL-S document. Estimate release date: July 31, 2004

   MWSAF (METEOR-S Web Service Annotation Framework): This tool can be used to do semi-automatic annotations. This tool takes in a WSDL file which has to be annotated and given a set of ontologies; it can classify the best ontology for annotating the WSDL and can also suggest most appropriate annotations for the inputs, outputs and operations in the WSDL file. The ontology classification can be done using schema matching or by using machine learning techniques. The tool can currently produce annotated WSDL 1.1 file using extensible elements and is being extended to produce WSDL-S as output. Estimate release date: July 31, 2004
   For more information visit: http://lsdis.cs.uga.edu/Projects/METEOR-S/MWSAF/

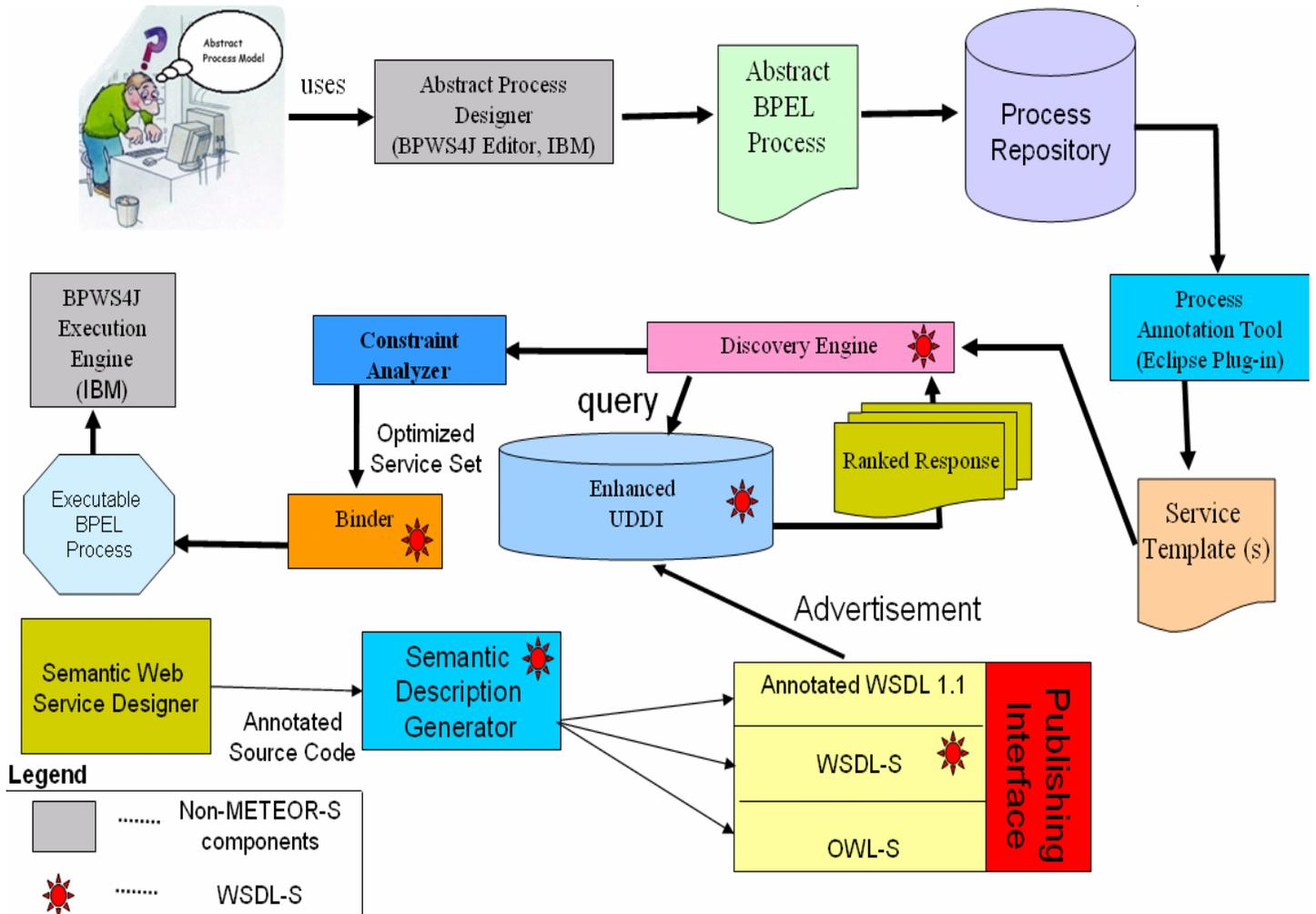## APPENDIX V - WSDL-S IN METEOR-S

Figure 2: WSDL-S support in METEOR-S components

Figure 2 shows the METEOR-S components and their working. METEOR-S will have extensive support for WSDL-S. The Semantic Web Service Designer produces annotated source code which can be converted to WSDL-S using the Semantic Description Generator. The Publishing Interface can publish WSDL-S files in the Enhanced UDDI. The Discovery Engine can search for WSDL-S web services from the UDDI and the Binder can use the information in WSDL-S file corresponding to the web service to extract more information required to bind a web service to an abstract process.