Abstract—Web services typically contain only syntactic information describing their interfaces. Due to the lack of semantic descriptions, service composition becomes a difficult task. To solve this problem, Web services can exploit the use of ontologies for the semantic definition of service's interface, thus facilitating the automation of discovering, publication, mediation, invocation, and composition of services. However, ontology languages, such as OWL-S, have constructs that are not easy to understand, even for Web developers, and the existing tools that support their use contain many details that make them difficult to manipulate. This paper presents a MDD tool called AutoWebS (Automatic Generation of Semantic Web Services) to develop OWL-S semantic Web services from UML models. AutoWebS offers an environment that provides many features required to model, implement, compile, and deploy semantic Web services.

Keywords—Model Driven Development; OWL; UML; Web Service; Semantic Web

I. INTRODUCTION

The semantic Web services (SWS) [1] encompasses the semantic definition of services through the use of ontologies. Ontologies provide a semantic description of a Web service (WS) that is computationally interpretable by combining the concepts defined using the ontology with the elements of the WS syntactically described in WSDL [2]. The development of SWS is typically divided into two stages: (i) WS creation, and (ii) WS ontology creation. There are languages that allow the description of WS using ontologies, such as OWL-S (Ontology Language for Web Services) [3]. The ontology languages have different syntaxes, a very extensive vocabulary, and most of them are based on first order logic.

Existing tools that support the use of ontology languages do not offer mechanisms to abstract their syntaxes, thus hampering their use [4]. The adoption of semantic descriptions of WS is hindered by the limitations of the tools and the fact that creating an ontology is a difficult and time-consuming task [5]. In order to facilitate the use of SWS it is necessary to abstract specific details for each semantic description language.

A way to guide the development of tools that meet this purpose is to define the essential requirements for such tools. In this sense, [6] proposed some essential requirements for the development of tools to compose WS. These requirements can be adapted to guide the development of a high level tool that provides an easy way to create atomic SWS. The requirements should define the need of abstracting the underlying technologies used in the development of SWS. Moreover, it is necessary to automate the generation of code artifacts, since it is expected to abstract the languages involved in the creation of SWS. The requirements should also specify the need of integrating functionalities to allow the creation of SWS without the need of accessing external resources or tools. This is essential to reduce the development time, to avoid errors and possible conflicts that arise when using different tools/applications such as conflicts of languages versions.

In this context, Model Driven Development (MDD) [7] is useful to manage the inherent complexity in the use of ontologies to specify SWS. MDD is a software development approach that focuses on creating models instead of program code, allowing separation of concerns between specification and implementation. Thus, a MDD approach can abstract away the underlying technologies of WS through models.

In the MDD approach, UML profiles can be used to create models that provide a higher abstraction level of the underlying technologies of SWS. A UML profile consists of a collection of extensions that customize UML for a specific domain and, models created by UML profiles are valid UML models, which can be created using the same tools for UML modeling. UML and ontology specification languages have some overlaps and similarities. For example, both languages use classes, associations between classes, class properties, generalizations, and data types for structural representation of a software system. Such similarities make it possible to make some elements of ontology specification languages in elements of the UML model [8]. Other elements of the ontology specification languages that do not directly correspond to the primitive elements of UML can be represented by using UML profiles. Thus, UML profiles can be used for specifying SWS, since they are capable to represent ontologies and WS interface.

This paper proposes a UML profile for the SWS and presents AutoWebS (Automatic Generation of Semantic Web Services), a MDD tool to create SWS. AutoWebS offers a graphical environment, which can be used to graphically represent OWL (Web Ontology Language) [9] ontologies as UML models, using the elements defined in a UML profile. Thus, it avoids that developers directly deal with the OWL language syntax by supporting the definition of SWS interface via UML models. This tool enables to: (i) specify the WS interface, i.e., model the inputs and outputs of each WS operation, (ii) perform semantic annotations, linking the inputs and outputs of operations with the elements of an ontology, (iii) automatically create the OWL-S ontology that contains the semantic description of WS, (iv) automatically generate the skeleton code for the WS, (v) extend its functionality, for example, including support for another semantic description language. The tool is implemented as a plugin of the Eclipse platform and uses EMF (Eclipse Modeling Framework) [10] for the specification of the OWL and OWL-S languages metamodels. AutoWebS uses the Axis2 middleware [11] for
generating WS and it also uses syntax validators for OWL-S ontologies.

This paper is organized as follows. Section 2 presents the requirements for a tool to create SWS, an overview of AutoWebS, and a motivating example. Section 3 presents the implementation details. Section 4 presents the results of a controlled experiment that evaluates AutoWebS compared to an application suite composed by the OWL-S Editor [12] and the Axis2 Eclipse plugin. Section 5 presents related work. Section 6 contains the conclusions and future perspectives.

II. AutoWebS

This section presents the essential requirements for a tool to support the creation of SWS, details of the AutoWebS, and a motivating example.

A. Requirements

Before the development of AutoWebS, a set of essential requirements for a tool aimed at creating SWS have been established. The requirements are intended to specify the need of abstracting specific details and the syntax of semantic description languages used in the creation of the semantic description of WS. The considered requirements are: (i) R0 To provide a mechanism for a user to model the SWS interface without the need to have a deep knowledge about Web technologies; (ii) R1 To perform the automatic generation of some WS source code; (iii) R2 To perform the automatic generation of the WS semantic description in a semantic language; (iv) R3 To allow the use of pre-existing ontologies to create semantic description of WS through the interconnection of the concepts defined in these ontologies with the elements of the WS; (v) R4 To provide a development environment that integrates all the features required to create a SWS, without the need to use external tools or resources; (vi) R5 To generate syntactically correct code artifacts. The generated code should be readable, executable, and meet the W3C specification for WS and SWS; (vii) R6 To allow the manutenability e.g., allow the insertion of new features without the need of performing major structural modifications in the tool.

R0 requirement can be achieved using UML models to specify the interface of the SWS, and to abstract the underlying technologies used to develop SWS. R1 and R2 requirements are related to automating the generation of code artifacts. In this work we adopt the OWL language because this work is part of a wider project which uses OWL. R3 requirement is concerned with the use of pre-existing ontologies to create semantic description of WS. This issue contributes to promote interoperability since it enables the use of existing ontologies that can be widely known and available on the Internet.

R4 requirement is essential to reduce the time of developing SWS and avoid possible conflicts that arise when different tools are used to build a SWS. For example, when using a tool to create the domain ontology that adopts a particular version of the ontology specification language that is not supported by other tool used to create the WS ontology. R5 aims to ensure the correctness of code artifacts generated by the tool. R6 is concerned with the insertion of new features or update of the current features, such as the upgrade to a newer version of the semantic language used to describe WS.

B. Tool Overview

AutoWebS provides an environment that integrates various functionalities used to create SWS (R4 and R6). The use of AutoWebS, as shown in Figure 1, consists of three main activities: (a) import domain ontology, (b) design the WS, and (c) generate the OWL-S ontology and WS. The tool requires as input OWL ontologies (R3) and produces as outputs OWL-S ontologies (R2) and the source code of the WS in Java (R1).

Figure 1. Overview of the AutoWebS

In the first activity, (a), the tool maps the elements of the OWL ontology to UML elements, and the result is a UML model (class diagram) that represents the OWL ontology. The UML profile customizes the model using stereotypes, constraints and tagged values, which are attached to UML model elements. They define the semantics of the UML model elements and make it possible to add additional information such as properties that define the port where are the WS, and its endPoint. In the (b) activity, the user works at the modeling level rather than on the direct manipulation of ontology languages. To create UML models that specify the SWS interface, the user relies on stereotypes, constraints and tagged values defined in the UML profile. From the perspective of the SWS designer, the tool resembles an UML class diagram editor (R0). In the environment provided by AutoWebS it is possible to create UML models that specify the SWS interface by using elements of the UML class diagram. In the (c) activity, the UML model that specifies the SWS interface is the input to a set of model-to-model and model-to-text transformations that automatically generate the semantic description of the WS in OWL-S ontology (R2) and build a WS project to the Eclipse IDE (R1). The created WS project contains: (i) the WSDL document, (ii) the descriptor of the WS, (iii) the classes that compose the SOAP communication infrastructure, and (iv) the Ant [13] script that automates the tasks of compiling and packaging the WS. To ensure that the semantic description of the WS is valid, the following validators are used (R5): the RDF validator; the OWL validator; and the OWL-S validator available in the OWL-S API.

C. Motivating Example

For the purpose of illustration of how to create a SWS, consider the WS Barnes & Noble Price Finder [http://www.mindswap.org/2004/owl-s/services.shtml] that returns the price of a book as advertised in Barnes and Nobles Web site given the ISBN Number. This WS has the following operation:


There are several technologies that can be used to implement this WS and also different ways to structure its inputs and outputs. For example, we can represent the input Book as a class that contains the attributes title and author; or the input can only be a String. Abstract types are elements specified within WSDL documents used to characterize the inputs and outputs of the WS.

Figure 2 illustrates the declaration of the abstract types for the Barnes & Noble Price Finder WS. The GetBNQuoteSoap-In and
GetBNQuoteSoapOut elements (lines 20 and 23) specify the request and response messages of the WS. Such messages are defined by GetBNQuote and GetBNQuote-Response Parts elements (lines 3 and 10), which provides a logical description of the message through XML Schemas.

```xml
<wsdl:types>
  <xsd:schema elementFormDefault="qualified">
    <xsd:element name="GetBNQuote">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element minOccurs="0" maxOccurs="1" name="sISBN" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="GetBNQuoteResponse">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element minOccurs="0" maxOccurs="1" name="GetBNQuoteResponseResult" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
</wsdl:types>
```

Figure 2. WSDL Document

In order to semantically describe this WS, the concepts defined in BibTex (http://purl.org/net/nkouf/hs/bibtex) and Concepts OWL ontologies may be used. The BibTex ontology formally defines the concepts and their relationships in the domain of the reference management software for formatting lists of references. The Concepts ontology defines the concept of price. In the BibTex ontology, Book is an OWL class that formally defines the concept of a book and in the Concepts ontology there is the Price class that defines price.

The OWL-S ontology that semantically describes the WS needs to model the GetBNQuote operation as an entity that exchanges semantic data serialized in XML messages and associates the input and output with elements of the BibTex and Concepts ontologies. The WS input should be associated with the Book class and the WS output should be associated with the Price class. The OWL-S ontology must also specify how the receiver can interpret the XML message and send it back into the semantic data and vice-versa. The transformations of the inputs and outputs are performed by XSL Transformations.

```xml
<owl:NamedIndividual rdf:about="#bibtexMyBook">
  <rdfs:label rdf:resource="#bibtexBook"/>
  <hasPrice rdf:datatype="xsd:string">1334</hasPrice>
  <hasYear rdf:datatype="xsd:string">2012</hasYear>
  <hasAbstract rdf:datatype="xsd:string">Abstract</hasAbstract>
  <hasAuthor rdf:datatype="xsd:string">Author</hasAuthor>
  <hasLocation rdf:datatype="xsd:string">Location</hasLocation>
  <hasPublisher rdf:datatype="xsd:string">Publisher</hasPublisher>
  <title rdf:datatype="xsd:string">Title</title>
</owl:NamedIndividual>
```

Figure 3. Instance of the OWL class Book

The process of manually creating an OWL-S ontology is an error prone task that consumes time and effort. For example, the OWL Book class used to semantically describe the WS input has several properties, among them hasPublisher, hasTitle, hasYear, humanCreator, and hasISBN. The structure of the OWL Book class and consequently an instance of this class is quite different from the WS input (String). Figure 3 shows an instance of the OWL Book class, which may be a hypothetical input of the WS.

This example shows some high-level activities to create a WS. The produced artifacts, such as the syntactic description of the WS (WSDL document) and the WS ontology are difficult to perform by hand. This example shows the real need for tools to create SWS.

III. IMPLEMENTATION DETAILS

This section presents the implementation details of AutoWebS. It is composed of: (i) an UML editor, (ii) a mechanism to import ontologies, and (iii) a mechanism for the automatic generation of SWS descriptions and some source code from a UML model (R4). AutoWebS also uses other plugins of the Eclipse platform: Axis2, Papyrus UML editor [14] and Ant.

AutoWebS uses the Papyrus UML graphical editor that supports UML profiles and allows creating custom editors based on UML standards. The QVTo plugin is used to perform the following model-to-model transformations: (i) from an OWL ontology (in OWL/XML format) to an EMF model (equivalent to the OWL metamodel), (ii) from an OWL model to a UML model; and (iii) from an UML model to an OWL-S model. The OWL-S document is created from OWL-S model using the code generator Acceleo. The source code of the WS is created from the UML model (R1) using UML to Java Generator plugin.

AutoWebS extends the functionality of Axis2 to enable the generation of source code and also to allow the creation of WSs projects to the Eclipse platform (R0). It also offers some facilities for compiling, packaging, and deploying WS. All features offered by this component are accessible via buttons or menus in our tool. New functionalities can be integrated into AutoWebS with the insertion of new plugins (R6) since the infrastructure offered by Eclipse enables the development of modular applications as plugins. Furthermore, the MDD approach implemented by AutoWebS allows the insertion of a new semantic description language. For this purpose, it is necessary to create a metamodel to such language and to create the model-to-model and model-to-text transformations.

A. UML Profile

The UML profile (Figure 4) defined and implemented by AutoWebS has two main purposes: (i) to allow the partial representation of OWL ontologies as UML models, and (ii) to represent meta-information about the WS in UML models. We chose to propose a new UML profile for OWL because the AutoWebS represent only the necessary OWL elements to specify the SWS instead of the full OWL ontology. In this profile the owl:ontology stereotype is applied to the definition of an UML Package and corresponds to the Ontology OWL element. This stereotype defines the ontology domain and contains information about the version of the ontology, author, comments, and namespaces’ declarations. An OWL ontology can import concepts from other ontologies. The owl:imports stereotype enables the relationship with other ontologies.

The owl:Class stereotype represents a concept that has been modeled as an OWL class. The OWL properties (DatatypeProperty and ObjectProperty) are mapped to attributes of a UML class with the owl:Class stereotype. In these attributes are applied the owl:DatatypeProperty and owl:ObjectProperty stereotypes. The rdfs:subClassOf stereotype is applied to the UML.
Generalization metaclass and represents OWL classes generalizations.

The Text-description and Category stereotypes extend the UML Comment metaclass and enable the textual description of the SWS and the association with a category. The Pre-condition and Effect stereotypes extend the UML Constraint metaclass in order to specify constraints which must be satisfied for the WS to properly run. In OWL-S preconditions and effects elements are represented as logical formulas and can be expressed with languages whose standard encoding is XML or literal strings. In the current version of the AutoWebS, logical formulas are set as UML Constraint.

The SemanticWebService stereotype is applied to an UML interface, which defines the WS operations (modeled as methods). This stereotype has the following properties: target-Namespace that defines the namespace used in the WSDL document; URI WSDL that determines the WS URI; Web-Service Documentation that serves for documentation purposes; servicePort specifies the port on which the WS responds; and the endPoint determines the type of endpoint used for end-to-end communication.

The methods defined in the interface stereotyped as SemanticWebService, which represent the operations of the WS are stereotyped as SWSOperation. The input and output of the operations can be defined as UML classes, UML primitive types or classes from the OWL ontology, which are represented in the UML model by owl:Class stereotype.

In our proposed MDD approach the import process of the OWL ontology is implemented in two phases. In the first phase a set of QVT transformations transform an OWL ontology in an OWL model (according to the OWL metamodel). In the second phase, some OWL elements of the OWL model are mapped to UML elements, so that at the end of the two phases the outcome is an UML model (UML class diagram) that need be manually extended to incorporate the WS definitions. In the resulting UML model, the UML profile provides additional meta-information inherent to the imported ontology and of the context of SWS. The UML model can be used to model the interface of the SWS. For example, the UML model includes properties that define the port on which the WS is listening, endPoint, and namespaces.

B. MDD Approach

AutoWebS implements a MDD approach, illustrated in Figure 5, in order to meet the R0, R1, R2, and R3 requirements. This strategy allows separation of concerns between the specification and implementation, and provides a high abstraction level of the OWL language.

From the UML model that describes the SWS interface, the tool automatically generate the WS project to Eclipse IDE (R1), and the OWL-S document. AutoWebS exports the graphical representation of the UML model to an XMI document, and from the XMI document that artifacts are created. To create an OWL-S document, AutoWebS uses an OWL-S metamodel and a set of QVT transformation rules that are applied to the UML model to automatically generate the OWL-S model (according to the OWL-S metamodel). From the OWL-S model, AutoWebS uses an Acceleo model-to-text transformation that transforming the OWL-S model into the OWL-S document. The source code in Java, which composes the WS project to Eclipse IDE, is generated from the UML model by the model-to-text transformation “UML to Java”. The WS operations represented through public methods in UML interface are mapped to methods in Java Interface, and they are used to generate the source code of the WS (each public method represents an WS operation). AutoWebS calls the API Axis2 to perform the following tasks: (i) to create the WSDL document associated with the WS; (ii) to generate the source code of the WS, i.e. the code artifacts that make up the SOAP infrastructure of the WS; (iii) to create the build.xml Ant script.

The main code artifacts generated are: i) the services.xml deployment descriptor, which contains the runtime configuration of the WS, ii) the MessageReceiver, which is responsible for the end-to-end communication, iii) Skeletons that implement the
protocol used for the transmission of messages, and iv) build.xml, which describes the build process and project dependencies with third-party APIs. The build.xml file is used by the Apache Ant to automate the building process and packaging the WS. All artifacts and the WSDL document are attached in a project for the Eclipse platform, so the user can develop the business rules of the WS. After the implementation of the WS business rules, the user can use the features offered by AutoWebS to build and deploy the WS.

IV. CREATING A SEMANTIC WEB SERVICE

This section shows how AutoWebS can be used to create SWS, demonstrating the process of creating the Barnes & Noble Price Finder SWS. The demonstration encompasses four steps: (i) to import the BibTeX and Concepts OWL ontologies; (ii) to model the SWS interface; (iii) to trigger the generation of the OWL-S ontology, and the source code of the WS; and (iv) to implement the WS business rules. Steps (i) and (iii) are automated by AutoWebS and the user only needs to manually perform the steps (ii) and (iv).

Step (i) imports the BibTeX and Concepts OWL ontologies, which contains the Book class that can be mapped to a UML class. The OWL Book class is a subclass of Entry and contains the properties: hasTitle, humanCreator, hasPublisher, and hasYear. The OWL Book class is mapped to UML as a packageElement element with uml:Class type and Book name. The UML Book class is a generalization of the UML class Entry. The OWL property classes are mapped to the ownedAttribute XMI element, keeping their names. The values are mapped to the UML primitive types.

The Step (ii) consists of adjusting the UML model. In this activity the user can insert or remove elements in the UML model. Figure 6 illustrates an UML model that contains the interface stereotyped with SemanticWebService. This UML model defines the BNPrice SWS. In this SWS the GetBNquote operation receives as input a Book element of the BibTex ontology and returns its price as a Price element of the Concepts ontology. The Step (iii) models the SWS interface. This step consists of defines the interface of the WS and triggering the mechanism of automatic generation of OWL-S file and WS project to the Eclipse platform. In this step the transformations presented in Section 3.3 are performed. The Step (iv) consists in the implementation of the WS business rules. After implementing the WS business rules, the next step consists in using the functionality of AutoWebS to build and deploy the WS. The WS project is built using Apache Ant's buildfile (build.xml). The result of the building process is a file with the .aar extension. To deploy this WS, AutoWebS makes a copy of the file with the .aar extension into the Web Container that has installed the Axis2 runtime.

![Figure 6. UML model to the semantic web service BNPrice.](Image)

V. ANALYSIS

For purposes of analysis, we have conducted a controlled experiment [15] that evaluated AutoWebS and an application suite comprised by the OWL-S Editor and the Axis2 Eclipse plugin, in the activities for the creation of SWS. The choice of OWL-S Editor and Axis2 Eclipse plugin was motivated by the fact that together they have some of the functionalities of AutoWebS and are used to create SWS. All the details of the experiment (hypotheses, methods and results) are available at www.consiste.dimap.ufrn.br/projetos/autowebs. We applied eight replicas of the experiment and created two SWS for each semantic WS project, each one created by a different tool. The results were analyzed with the Wilcoxon nonparametric statistical test [16]. Table I shows the times, in minutes, required for developing each SWS and the number of errors or inconsistencies in each OWL-S ontology.

<table>
<thead>
<tr>
<th></th>
<th>AutoWebS</th>
<th>Application suite</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>error</td>
</tr>
<tr>
<td>OilMonitor 1</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>OilMonitor 2</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>Book Finder</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>Zip Code Finder</td>
<td>5</td>
<td>21</td>
</tr>
<tr>
<td>LatLongFinder</td>
<td>8</td>
<td>25</td>
</tr>
<tr>
<td>Barnes &amp; Nobles</td>
<td>5</td>
<td>21</td>
</tr>
<tr>
<td>BabelFish Translator</td>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>Currency Converter</td>
<td>5</td>
<td>14</td>
</tr>
</tbody>
</table>

The WS ontologies created by AutoWebS do not have errors or inconsistencies and in all OWL-S ontologies created by the application suite, errors or inconsistencies were found.

Figure 7 illustrates the time consumed by the participants to develop each SWS. We can see a difference between the times when using AutoWebS and OWL-S Editor combined with Axis2. In all cases, the development time using AutoWebS was shorter than the time when using the application suite.

![Figure 7. Time to develop each semantic web service (SWS)](Image)

Based on data contained in Table 1, we can conclude that for all SWS the values for the metrics "time" and "errors" for AutoWebS were smaller than or equal to the application suite. All data from the controlled experiment at available in www.consiste.dimap.ufrn.br/projetos/autowebs.

VI. RELATED WORK

Some MDD approaches to create OWL-S ontology have been proposed in the literature. (i) ASSAM [17] is a tool that provides a graphical interface based on views. It uses a semi-automatic process where the user defines the mappings between concepts of an OWL and the Message elements of the WSDL document. This tool uses a learning algorithm that when an user defines the mappings, it presents some suggestions of the ontology classes that can be associated with the WSDL elements and (ii) CODE [18] is
a plugin for the Eclipse IDE. CODE provides a converter called WSDL2OWL-S to generate the semantic description of WSs in OWL-S. The tool has four editors that enable to edit the OWL-S sub-ontologies. The editors are based on forms that must be filled with data from the WS ontology. The aforementioned tools apply different approaches. Table 2 presents a comparison of them with AutoWebS, considering the requirements for a high-level abstraction tool for creating SWS presented in Section 2.1. “Y” means that the tool fully meets the correspondent requirement, “N” means that the tool does not meet the requirement, and “P” means that the tool partially meets the requirement.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>P</td>
<td>N</td>
</tr>
<tr>
<td>(ii)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>P</td>
<td>P</td>
<td>Y</td>
</tr>
</tbody>
</table>

AutoWebS

As shown in Table 2, ASSAM does not fulﬁl the R0, since it requires an extensive knowledge about XML Schemas and OWL. Moreover, the manual association between ontology concepts and the WSDL elements may be diﬃcult to be performed, since the number of possible combinations of mappings may be large. This tool is not integrated into a development environment and it does not provide functionality for creating the source code of the WS, and the WSDL document. Thus, ASSAM is not in compliance with the R1 and R4. The architecture of the tool does not support the integration of new functionalities, therefore it does not meet the R6.

The CODE tool has no mechanism to import concepts deﬁned in an ontology and therefore it does not meet the R3 and R4. Furthermore, the OWL-S ontology generated by the WSDL2OWL-S converter is incomplete and requires manual processing to supplement the ServiceProfile and ServiceModel sub-ontologies. For this reason, this tool partially meets the R5 requirement. As CODE uses the Eclipse platform, it is possible to integrate new modules that provide new functionality. Thus, it meets the R6. The Java2WSDL and WSDL2OWL generators provide some abstraction of the underlying technologies used to develop the SWS, thereby meeting the R0.

VII. Conclusions

This paper presented a MDD tool to make the development of SWS more intuitive and easy. The creation of atomic SWS (WS and its ontology) is still a recurrent problem that deserves attention because most tools do not support all the development stages of SWS. The few tools that support all stages fails at some point, either as the quality of the artifacts generated as, for example, the incompleteness of OWL-S ontology created from WSDL2OWLS converters, or the layout and presentation of the tools that requires a great knowledge about the SWS technologies. The approach implemented by AutoWebS allows developers to focus their efforts on creating models instead on writing source code. Additionally, the fact that the models created from UML proﬁles are valid UML models and due to extensive use of UML as modeling language, this approach makes AutoWebS accessible to a wider audience. AutoWebS automatically creates OWL-S ontologies and WS projects from UML models. Using the tool it is possible to model SWS in a platform-independent way through UML models.

AutoWebS has some limitations that do not prevent its use. The current version of the algorithm for creating XSLT scripts is limited, and it is only able to create XSLT scripts to ontology concepts which have properties deﬁned as primitive types or other concepts deﬁned as OWL classes. For more complex cases, for example, lists, our algorithm is unable to correctly make its interpretation and therefore cannot generate the XSLT script. As a future work we intend to evolve AutoWebS by creating a GUI for specifying conditions (Preconditions, Results, and Effects) used in WS ontologies.

References