iSemServ: A model-driven approach to developing semantic web services

Jabu Mtsweni*†, Elmarie Biermann*, Laurette Pretorius‡

*School of Computing, University of South Africa, Florida Campus, Johannesburg, South Africa
†Defense, Peace, Safety and Security, Council of Scientific and Industrial Research, Pretoria, South Africa
‡School of Interdisciplinary Research and Graduate Studies, University of South Africa, Pretoria, South Africa

ABSTRACT

The benefits of incorporating Semantic Web Services in web applications are well documented. However, both the real-world implementation and adoption of these services are still rather limited in practice. This is despite the promises that extend syntactic Web services with capabilities such as automatic service discovery, composition, and execution. Some of the barriers to the real-world implementation are the complexities and tool support related to the development of Semantic Web Services. In this article, the main challenge that is addressed is the tight coupling of existing Semantic Web Services (SWS) development platforms to specific semantic description languages and service description languages, which unintentionally lead to unbending service development environments. The main contribution in this article is therefore a model-driven approach called *iSemServ* that exploits mature technologies, such as UML, and model-transformation techniques for simplifying and semi-automating the development of SWS using description languages of choice, such as Web Ontology Language for Services (OWL-S) and Web Application Description Language (WADL). A design science research methodology was employed in conducting the study. The suggested approach was practically implemented as an Eclipse plug-in and evaluated based on a real-world use case scenario and comparative analysis of related solutions. The evaluation results show that our proposed solution is relevant and appropriate in aiding the semi-automatic development of SWS, albeit with a number of limitations that could be addressed by extending the proposed practical solution.

KEYWORDS: iSemServ, Semantic Web, Web Services, Semantic Web Services, Ontologies, Web engineering, Model-driven architecture

CATEGORIES: D.2, D.2.11

1 INTRODUCTION

The benefits of employing Semantic Web Services (SWS) in web applications are well documented in academia and industry [1] [2] [3] [4]. Some of the benefits of utilising SWS include:

- improved representation, sharing, searching, reasoning and reuse of data and services on the Web [5], and
- automation of various web-based tasks, such as service discovery, selection, composition, choreography, orchestration, and execution [5] [6].

Nevertheless, the real-world implementation and adoption of SWS has remained limited to date. Various major challenges that are contributing to this lack of implementation and usage have been identified [1] [7] [8] [9]. Some common issues include the lack of effective tools [4], non-integration of SWS technologies into existing technologies [10], high costs of adopting service-oriented architectures [11], steep learning curves, high complexity of prominent heavy-weight semantic models, and concerns over agreement in semantic modelling standards [6].

Preliminary investigations suggest that existing approaches for aiding the development of SWS are fragmented and tightly coupled to specific semantic description languages and service description languages, also referred to as service architectural styles, leading to standard-dependent and restrictive development environments. This further leads to undesirable consequences, such as prolonged service development processes and additional implementation costs [6] [11].

New approaches that support multiple semantic description languages and architectural styles of choice are indeed essential towards facilitating, promoting, and semi-automating the development of SWS. Therefore, the main contribution of this article is the presentation and description of a model-driven approach for designing and developing SWS using semantic description languages and service description languages of choice.

A design science research methodology [12] was followed to design and develop the proposed model-driven approach. The proposed and evaluated solution

Email: Jabu Mtsweni `mtswenij@gmail.com`, Elmarie Biermann `bierman@xsinet.co.za`, Laurette Pretorius `pretol@unisa.ac.za`

is based on the principles of multiple description languages support and complexity hiding through model-driven techniques, automatic code generation through dynamic templates, and extensibility to accommodate new description languages.

The remaining sections of this article are structured as follows. Firstly, detailed background information relating to SWS and their essential components is provided. This is followed by explaining the rationale of the research methodology used for devising and evaluating the proposed model-driven approach. Related existing solutions that attempt to address the SWS development challenges are then discussed. Next, the delineation of the essential design requirements that serves as key building blocks for the proposed model-driven approach are discussed. The approach, termed iSemServ, for building SWS is then presented and described. The implementation and evaluation results of the suggested solution using a well-defined use case scenario and comparative analysis are reported and the article is concluded by highlighting possible future research.

2 BACKGROUND

The fundamental elements that comprise SWS are mainly web services, syntactic descriptions, semantic descriptions, and ontologies. These discrete elements demonstrate that the development of SWS is not a trivial process. The understanding and discussion of these elements is therefore essential, as they also show the complexities and diversities involved when Information System developers design and implement SWS.

By definition, SWS originate from the integration of syntactic Web Services (WS) and the Semantic Web (SW), which is an extension of the current Web [10] [13]. The term 'syntactic' is used in the context of this article to distinguish the mostly HTML-based World Wide Web from the Semantic Web, in which web pages carry information that can be read and understood by both machines and humans in a systematic way. The leveraging of syntactic WS with semantic descriptions [14] is preferred mainly for automating business processes on the Web [3] [15], thus minimizing human intervention in performing tedious Web activities, such as searching and discovery of large amounts of information.

The main objective of SWS is to enable machine-processable and machine-interpretable services with limited human intervention by using ontologies and semantic descriptions to unambiguously describe both functional and non-functional aspects of services.

The following subsections discuss the salient components of SWS.

2.1 Semantic descriptions

From the service-orientation perspective, semantic descriptions are meant to formally describe the core capabilities of a service, and this includes the procedure on how these capabilities could be executed (i.e. service behaviour), accessed, aggregated, and consumed by users such as software agents and humans [2].

Within the SWS domain, a number of heavy-weight and light-weight models on how to formulate semantic descriptions that could be linked to syntactic descriptions, using standards such as Web Service Description Language (WSDL) [16] and Web Application Description Language (WADL) [17], are emerging. The main prominent semantic models to date are Web Ontology Language for Services (OWL-S) [18] and Web Service Modeling Ontology (WSMO) [19].

The goal of heavyweight semantic models is to exploit commonly agreed upon vocabularies as domain and service ontologies to represent and describe different aspects of Web services (WS) separately from syntactic descriptions. OWL-S is one of the first heavy-weight semantic models based on the Web Ontology Language (OWL). It provides a structure for defining semantic descriptions through the use of so-called service profiles that semantically describe what a service is capable of offering to prospective consumers.

The service model, using OWL domain ontologies, describes the specific behaviour of a service in terms of its input, output, pre-conditions, and effect, and service grounding describes how SWS could be automatically invoked and executed by humans and software agents.

Another conceptual model for semantically describing salient aspects of Web services, called WSMO, has also found prominence within the field of SWS. It is based on Web Service Modeling Language (WSML) [19]. It focuses on four elements that are important in semantically describing services:

'Ontologies' provide formal concepts that could be used by other WSMO elements

'Web Services'[1] describe the functional, non-functional, and behavioural aspects of a service

'Goals' semantically capture consumers' requests and could invoke service capabilities; and

'Mediators' are responsible for handling the incompatibilities and mismatches between terminologies used across different WSMO elements.

These semantic description specifications are considered to be efficient and expressive for realizing SWS. However, their manual use by developers for creating semantic descriptions [20] [21] are complex. Furthermore, current solutions for the development of SWS are generally not tailored to address or hide some of the complexities of semantic description specifications. Furthermore, most of the development environments are tightly coupled to specific description languages, which might not cater for the needs of developers familiar with other existing semantic and syntactic description languages.

Moving to more light-weight approaches may be one possible solution in eliminating some of the complexities inherent in languages such as OWL-S and WSMO, however, these light-weight approaches, such

[1]These are WSMO web services and not traditional web services.

as WSMO-Lite [22] are considered less expressive leading to hindrances in fully automating the various SWS activities (e.g. service composition).

2.2 Ontologies

Semantic descriptions are generally derived from ontologies, which are defined as formal representations of knowledge in a particular domain [23]. According to [10], ontologies are an essential requirement for the implementation of SWS, due to their "explicit declaration of knowledge", which makes it possible to automate various activities on the Web. Ontologies are core to the overall development of SWS. They are essential toward enabling automatic service discovery, composition, and execution as they make imparting semantic data to Web resources possible. Furthermore, ontologies are important in ensuring that information is clearly understandable to both humans and machines [10], by minimizing ambiguities in concepts and relations used in a specific domain.

Ontologies are often classified according to different types of categories such as top-level, domain, task, and application ontologies [24] [25]. In the context of this article, the focus is mainly on service ontologies, which fall into the category of application ontologies [25]. Domain ontologies are meant to capture common and shared knowledge about a specific domain (for example: health), whilst service ontologies (also referred to as semantic descriptions) focus on semantically describing the internal and external operations of Web services using domain ontologies.

In practice, ontologies are often developed using disparate ontological languages and tools [26]. However, in the semantic services domain, ontologies are commonly produced with the assistance of semantic models (e.g. WSMO). As highlighted in the introductory section, there are still concerns in the SWS domain regarding the standardization of semantic models. To date, there are no de facto standards for developing service ontologies. In addition, there are no commonly agreed-upon approaches on how to connect semantic descriptions with syntactic descriptions. As a result, existing semantic services platforms tend to support a specific semantic model and service architectural style (e.g. RPC-based services) [27]) and to ignore others.

The following subsection highlights background information on common syntactic descriptions languages, which are the foundation for semantic descriptions as discussed above.

2.3 Syntactic descriptions

The notion of traditional Web services (WS) has experienced significant uptake both in industry and academia over the past few years. Some of its successes could be attributed to the maturity of tools and development environments, and the backing of prominent IT companies, such as Oracle, Google and Microsoft. Similar to SWS, WS rely heavily on descriptions. In the case of WS, syntactic descriptions are the foundation, which generally focuses on describing capabilities of services without focusing on their intrinsic behaviour.

The most common syntactic description language, as highlighted in the previous section, that exists to date is the Web Service Description Language (WSDL), targeted at what is referred to as "Big Web Services" or SOAP-based and Web Application Description Language (WADL) meant for describing RESTful services. It should also be noted that WSDL has improved over time in that it can now also be used to describe RESTful services.

WSDL is mature and standardized, and is widely adopted. It exploits the XML language with standardized schemas to syntactically define and describe WS capabilities [28]. However, services described with WSDL lack semantic descriptions, which are essential for producing SWS. On the other hand, WADL, an XML-based description language, is beginning to be used successfully by RESTful services developers. It provides descriptions for web-based services and applications [17]. It is generally considered light-weight compared to WSDL, due to its reliance on open protocols [17]. In summary, different service architectural styles can be used for implementing WS, which are a foundation for SWS.

3 RELATED WORK

In this section we highlight some of the related studies that proposed different ways to address the challenges of designing and developing SWS.

One of the solutions that claims to be the first towards SWS engineering is called INFRAWEBS [7]. It focuses on constructing semantic descriptions for existing and new WS, and enables the integration of disparate components. INFRAWEBS is made up of different units (i.e. SWS creation, monitoring, selection, discovery, composition, and conversion), which are paramount to the actual development and implementation of SWS. However, INFRAWEBS suffers from being bound to a specific ontological language (i.e. WSMO) and service architectural style (i.e. SOAP-based services).

ODE-SWS, a SWS development environment [29], focuses on the design and development of SWS at a knowledge level. It does not put restrictions on the semantic description language that could be used as long as it is compatible with the WebODE framework, an ontology engineering workbench, responsible for exporting provided ontologies into other ontology languages [30]. However, it is limited in the sense that it only supports "Big Web Services" described using WSDL and does not attempt to hide some of the complexities that are inherent in different semantic description languages. Nevertheless, the design requirements used for the ODE-SWS approach are incorporated into our proposed solution.

Another research endeavour that is closely related to our work is a practical integrated development environment (IDE) called OWL-S IDE, and formally known as CMU's[2] OWL-S Development Environment (CODE) for developing, deploying, and consuming SWS [26]. OWL-S IDE adopts and extends existing WS tools (e.g. OWL-S editor) in order to support developers with the process of developing, deploying, and consuming semantic services [31]. It is embedded within the Eclipse environment, and is purely based on Java and OWL-S.

It follows both code-driven and model-driven methodologies in delivering SWS. This approach also supports various SWS development stages, such as discovery, invocation, and execution. However, it does not cater for other specifications, such as WSMO or WADL.

Lastly, a reverse engineering approach for building SWS is proposed by [15]. This solution opts for a model-driven approach. Existing syntactic descriptions in WSDL format are translated into UML class and activity diagrams. The generated diagrams are then manually annotated and converted into OWL-S descriptions. Similar to all the approaches discussed, this solution is tightly coupled to OWL-S descriptions. A criticism of this approach is that it also relies entirely on syntactic descriptions for generating semantic descriptions through reverse engineering, which could limit the expressiveness of intended SWS.

Based on the overview presented in this section, it is apparent that current SWS development platforms are restricted and in various cases tightly coupled to specific service architectural styles and semantic description models.

We may therefore conclude that since there are no common standards for defining semantic descriptions, it is appropriate and essential that emerging SWS development approaches should cater for diverse description languages and different semantic description models.

4 RESEARCH METHODOLOGY

The primary research method used to arrive at the proposed solution is the design science research (DSR) methodology [12] [32]. DSR is suitable as it is a systematic problem-solving method for producing relevant, new, and innovative Information Systems (IS) solutions within a specific domain. DSR is also relevant for addressing problems that have dynamic or variable requirements and other constraints [33]. The challenge of supporting SWS development using multiple description languages falls into this category.

According to [33], there are two positions that can be assumed for conducting design science research, the "paradigmatic-design" and the creation of Information Technology (IT) artefacts, for example, methods, models, instantiations, or a combination of these. In our context, we focus on the creation of an IT artefact.

The DSR guidelines used for arriving at the proposed solution are depicted in Figure 1. Firstly, we identified the relevant problem within the domain of SWS development using extensive literature review

[2]Carnegie Mellon University.

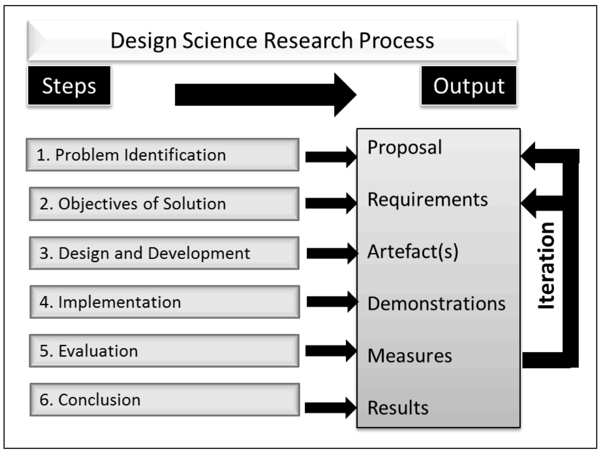


Figure 1: Steps in the Design Science Research Methodology (adapted from [32])

as discussed in the previous sections. This process resulted in a need for a new approach that could facilitate the semi-automatic development of SWS using diverse description languages.

Secondly, the objective of the solution, viz. the formulation of design requirements for guiding the construction of the proposed model-driven approach (referred to as an artefact in DSR [32]), was stated. In particular, we dissected the identified problem and features of the selected related solutions for building SWS.

Thirdly, the requirements gathering phase formed the basis for designing the new and innovative model-driven approach for facilitating the practical construction of SWS using description languages of choice. The approach is presented in a form of a process model, using a modelling-by-design strategy [34]. The modelling-by-design method is preferred, as it has been found to be appropriate for capturing the essential components of a complex system or process [34].

Fourthly, in the demonstration phase, the modelled approach was then practically implemented using mature and existing development methods and platforms, such as UML, Acceleo, and Eclipse. These methods and tools are discussed in the implementation section of this article.

Finally, once the proposed solution was implemented, it was then evaluated for utility, significance, and relevance using a practical use case scenario and comparative analysis approach (also referred to as static analysis in the design evaluation phase of DSR [32]). In this phase, the proposed model-driven solution was theoretically compared against other existing solutions using the identified design requirements. It should also be noted that DSR is iterative, allowing the artefact to be improved through various iterations as required. The results of the analysis are presented in the evaluation section.

5 SOLUTION OBJECTIVE: DESIGN REQUIREMENTS

In this section, we discuss the design requirements that form the foundation of the proposed solution. These requirements are itemized and explained as follows:

Model-driven. According to [29], the development of SWS needs to be carried out at the conceptual level using platform independent models that could be transformed to platform-specific models, and eventually to code. Thus, any solution that attempts to address the semi-automatic development of SWS using multiple languages needs to consider a model-driven approach. Modelling as a means of abstraction and automation is widely accepted in software and web engineering [35], especially for purposes of simplifying the development experience. In addition, model-driven approaches are considered efficient and effective in relation to developing "complete" service-based systems [31]. Models are also important for automatic code generation due to different levels of abstractions [36], thus maximizing complexity hiding and service engineering productivity. Our aim is to enable developers to model SWS by specifying their description languages of choice using modelling languages, such as UML. The proposed solution would then automatically translate the model into all the necessary elements of SWS based on the description language that is preferred by the developer.

Decoupling. This requirement ensures that the proposed solution promotes the separation of concerns as much as possible. Similar to WSMO elements, the proposed solution needs to support the definition of service descriptions, semantic descriptions, and ontologies in a language-independent manner. However, these elements still need to be aware of each other, and be easily integrated when needed. The ODE-SWS framework refers to this requirement as modular design, meaning the framework is composed of a set of independent but related modules [29].

Use of multiple description languages. Existing approaches as discussed tend to only accommodate one particular language for describing services syntactically and semantically. Those that claim to support language independence, such as the ODE-SWS framework tend to simply focus on semantic descriptions, and not focus on syntactic descriptions, which are core to the formation of SWS. In our approach, multiple description languages are accommodated as long as they are capable of describing various aspects of WS and SWS, such as functional, non-functional, technical and behavioural.

Complexity hiding. The proposed solution needs to support approaches that could aid developers in rapidly implementing SWS components and reusing existing ones. In addition, it should support the use of tools capable of reducing inherent complexities when implementing SWS and related applications [7].

Extensibility. The proposed approach needs to be extensible in a sense that new description languages are easily supported and integrated into the development environment as they become available.

Integrative. Developers of SWS need to be able to uniformly and cohesively perform all the activities (e.g. modelling, development, description, annotation, and others) of building SWS within one development environment. It is also asserted in [29] that any SWS development solutions need to be easily integrated with already existing and mature Web services de facto standards.

6 PROPOSED APPROACH

The proposed model-driven approach is called iSemServ (Intelligent Semantic Services). This is referred to as an artefact in DSR. It should also be noted that an extensive version of the proposed solution, which includes the support for developing intelligent SWS is presented in [9]. The approach as demonstrated in Figure 2 is presented as a multi-layered architecture, made up of three core layers, namely: services layer, semantics layer, and knowledge layer. The layers are derived from the core elements of SWS. Although the layers are interlinked, they are not dependent on each other for operation, but only for producing functional SWS. The iSemServ also offers the ability to start the development process from any layer as long as the service model is made available.

In the subsequent section a detailed description of the proposed iSemServ approach in terms of its core layers is provided.

6.1 Services layer

In general, the service development process begins with the concept of a raw service. That means the process begins at the requirements elicitation phase. Once all requirements to be satisfied by a concrete service are identified, the iSemServ approach could be used as an end-to-end approach to develop any type of SWS.

In the services layer, the initial step for developing SWS deals with service modelling facilitated by the Service Modeller module. This component satisfies the model-driven requirement. It also contributes towards addressing the challenge of building semantic services by using multiple description languages of choice. The multiple description languages requirement is addressed by the Service Modeller through the use of the iSemServ UML profile, which is discussed in the implementation section.

In the service modelling phase the envisaged SWS is represented in platform independent models. This step also promotes the decoupling of business logic and service logic as highlighted in the design requirements section.

Once a service model is available for specific SWS, the model could then be automatically transformed using defined Model2Code templates and transformation rules into code skeletons that represent the classes and operations relevant for service logic implementation. The automatic code generation process is necessary to satisfy the complexity hiding design requirement.

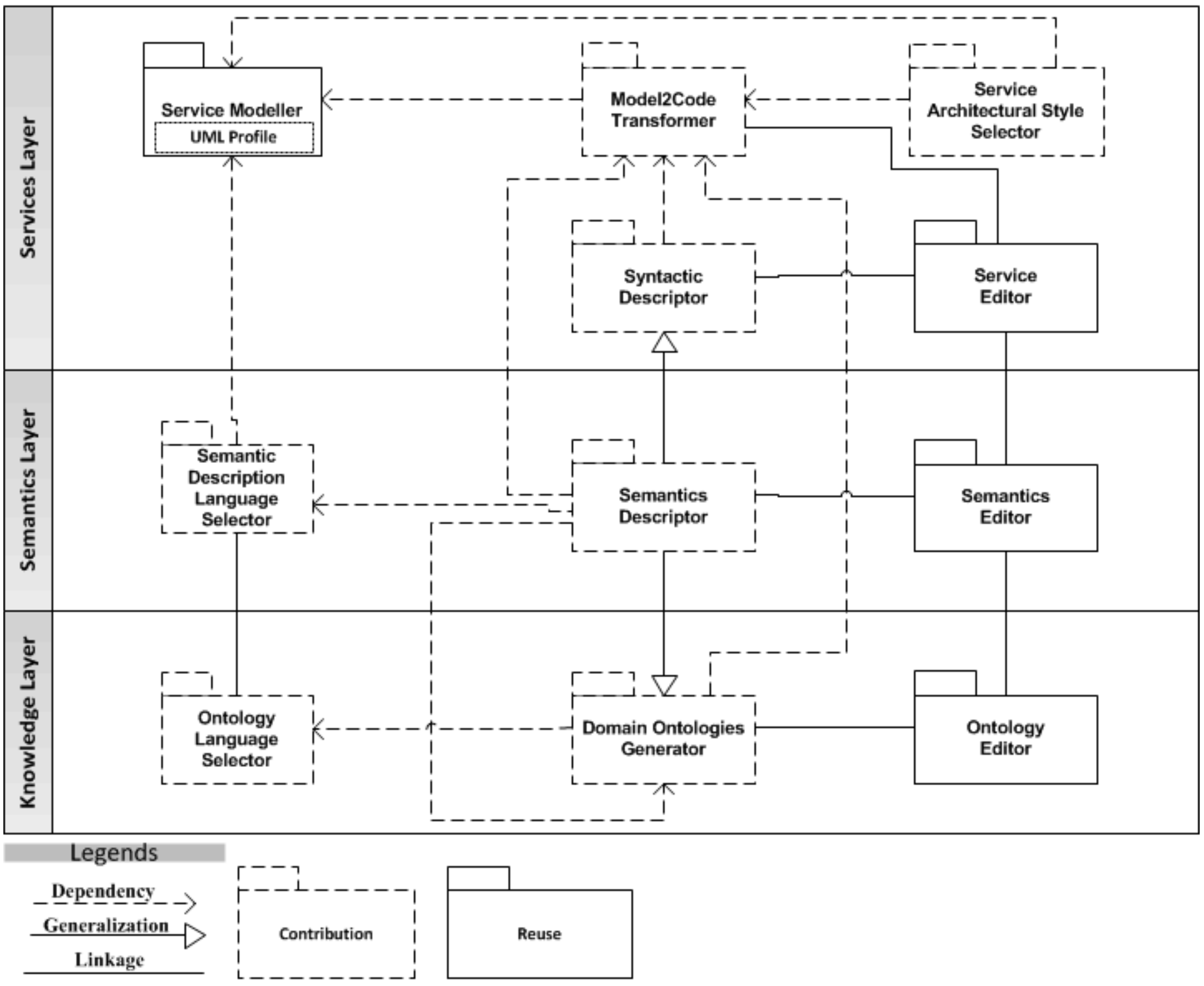


Figure 2: iSemServ model-driven approach

The code skeletons could be supplemented or edited by a service developer using any relevant programming editor. In this work, editors are integrated within the overall development environment as explained in the implementation section. This is in line with the design requirement of ensuring a unified SWS development environment. As illustrated in Figure 2, the iSemServ model supports the development of different types of Web services, made possible through the Service Architectural Style Selector. The selector depends on the transformation rules defined within the Model2Code transformer module, which were considered for addressing the extensibility and complexity hiding requirements.

Syntactic descriptions are automatically generated by the Syntactic Descriptor module based on the service model realized or reused within the Service Modeller module. The type of syntactic descriptions to be generated would depend on the annotations injected into the service model using the iSemServ UML Profile. For example, the developer could annotate the service model with WSDL stereotypes to indicate the preference to generate WSDL service descriptions.

In conforming to the decoupling requirement, at this layer once the syntactic descriptions are generated, and the service logic implemented, syntactic Web services are available for deployment, publication, and execution. However, these would be mere syntactic services without any semantic descriptions, thus not SWS.

6.2 Semantics layer

The semantics layer depends on the Service Modeller and Model2Code transformer modules for semi-automatically generating semantic descriptions based on the language chosen by the developer. It should be noted that the syntactic descriptions produced in the services layer could also be used as input to the semantics layer's Semantics Descriptor module as depicted in Figure 2. Nevertheless, this approach is not preferred as important semantic details might be lost when translating the service model to syntactic descriptions.

Due to a diverse group of semantic description languages that could be used to semantically describe Web services, the proposed solution provides the developer with the ability, through the use of the iSemServ UML profile, to choose the preferred semantic description language. The Semantic Description Language Selector module is capable of deciphering such a choice from the available service model in the services layer.

Depending on the selection, semantic descriptions could then be automatically generated using the Model2Code transformation rules. However, since the semantic descriptions generated by the Semantics Descriptor module might be incomplete due to incomplete service models, the developer is also provided with a Semantics Editor module to visualize, edit, augment, and validate the generated semantic descriptions. The semantic descriptions generated in this layer rely directly on domain ontologies delivered in the knowledge layer.

6.3 Knowledge layer

It is possible for developers to use one language for developing domain ontologies and another for defining semantic descriptions. In order to accommodate this possibility, the iSemServ solution includes the knowledge layer where the developer is offered the opportunity to choose a language for generating domain ontologies. The Ontology Language Selector, which depends on the service model, is one of these modules as shown in Figure 2. The Domain Ontologies Generator would then use the information about the language selected, available through the Ontology language Selector, to automatically generate domain ontologies. These could then be used by the Semantic Descriptor to produce relevant semantic descriptions.

The outputs of the knowledge and semantics layer are independent semantic descriptions and domain ontologies that describe services realized in the services layer. In this layer, when all the core elements are integrated, we then have a functional semantic web service.

The following section discusses the important choices we made regarding the implementation of the iSemServ model-driven approach.

7 IMPLEMENTATION: ISEMSERV

The iSemServ approach was implemented on the Eclipse platform, which encompasses a variety of reusable service engineering components. It must be noted that the proposed approach could be implemented using any other SOA-based platform. The decision to implement the framework using the Eclipse[3] environment was motivated by a number of factors and benefits, such as openness, wider support and community involvement, the availability of plug-ins, ensuring extensibility, and support of multiple programming and modelling languages.

Although the implementation exercise was meant to demonstrate the proof-of-concept, rather than a fully-fledged iSemServ model-driven platform, an effort was made to implement most of the salient features necessary for using multiple description languages in the development of SWS.

Adhering to the decoupling requirement, the implementation was realized in phases. Thus, each layer was implemented independently. Nevertheless, the completed implementation involved the integration of all the layers into one operational iSemServ Eclipse plug-in. The following subsections discuss the implementation details layer by layer.

[3]`http://www.eclipse.org`.

7.1 Services layer

The service modeller, which represents the core module responsible for capturing the internal and external properties of the identified services, was implemented by following the Model-Driven Architecture (MDE) as coined by the Object Management Group (OMG) [37]. In general, service models could be derived using any modelling language of choice. However, MDA compliant languages such as the Unified Modelling Language (UML) are encouraged by OMG.

Thus, for the implementation of the service modeller module, UML compliant models are preferred. This is mainly because of their wide spread use in industry and academia, and support for platform independency ensuring “portability, interoperability, extensibility and reusability through an architectural separation of concerns between the specification and implementation” [38].

Using the UML development kit integrated within the Eclipse platform, the service designer would capture the structure of services identified using UML class diagrams. The behaviour of identified services could also be captured using UML activity diagrams. Nonetheless, this was not implemented for this article.

For proper functioning of the service modeller module, the class diagrams capturing the properties of services need to be modelled following the novel iSemServ UML profile. In a nutshell, UML profiles are a group of custom keywords (i.e. stereotypes), data types and tag values that could be used to annotate and extend UML diagrams [15]. Moreover, the distinct stereotypes within the UML profile provide the flexibility to annotate the model in a manner that would promote different representations of the model, and in our case support the development of SWS using multiple description languages. It also is important to note that UML profiles are easily implementable using any UML compliant tool, and could be extended by adding new keywords and preferred rules.

Figure 3 shows the iSemServ UML profile implemented for the proposed solution, as rooted within a UML package. This is important for Java code generation, where classes are generally organized within a package. The key stereotypes in the profile are ≪RESTful≫ and ≪SOAP≫. These two stereotypes can only be applied to class diagrams. This means that any class diagram capturing the structure of a service could be annotated as ≪RESTful≫ or ≪SOAP≫, the two common Web services standards to date. The other stereotypes deriving from the main stereotypes are ≪WADL≫ and ≪WSDL≫, which enable the developer to decide on the syntactic descriptions to be generated from the service model.

In terms of the preferred semantic descriptions and domain ontologies, ≪WSMO≫ and ≪OWL-S≫ stereotypes are shown to demonstrate the plausibility of multiple description languages for the development

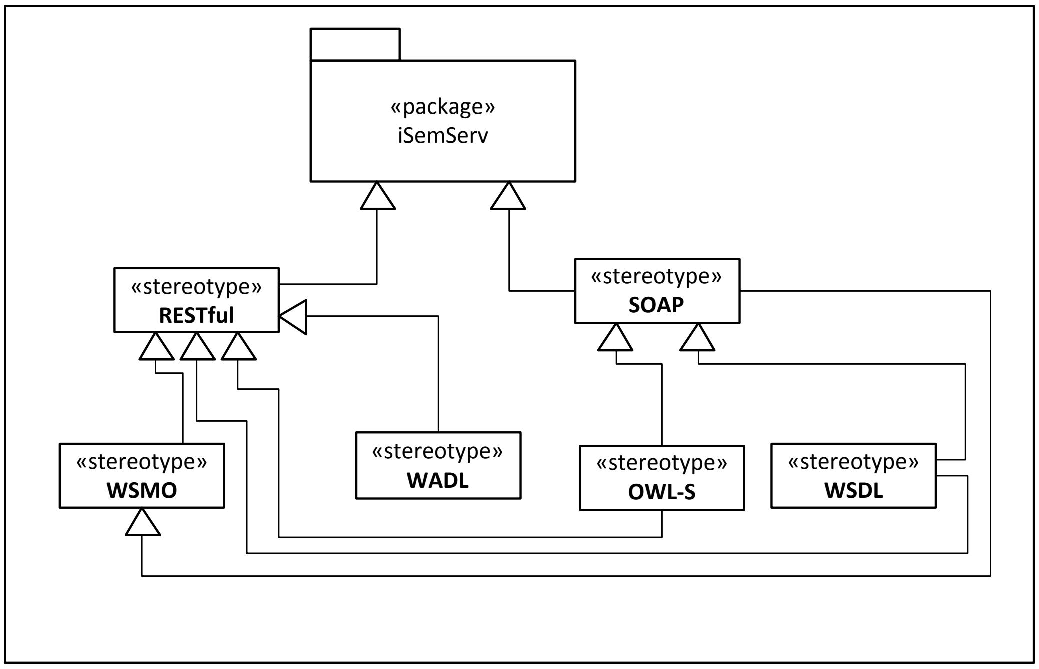


Figure 3: iSemServ UML profile

UML Class Diagram	Java Class (JAX-WS)
<<SOAP>> class	SOAP Service (@WebService)
Class Name	Service Name
Package Name (reversed)	TargetNameSpace
Owned Operation	@WebMethod
Package Name +Interface	EndpointInterface

Table 1: UML2SOAP mappings

of SWS. Additional keywords could also be added to accommodate other syntactic descriptions and lightweight semantic descriptions or annotation standards, such as WSDL-S [39].

The model2code transformer was implemented using a number of code transformation rules and templates defined solely for the proposed model. The rules and templates are based on the Acceleo platform integrated within Eclipse. In brief, Acceleo [40] is an open source model-to-text language (MTL) framework. It provides a flexible and simple environment for designing and developing a variety of code generators, using simple and standard templates. It is used for our proposed solution, primarily because its design principles are based on increasing software development productivity, satisfies the complexity hiding design requirement, and supports different types of high-level programming languages, such as Java that are commonly used for implementing Web services logic.

Transformation rules that were implemented include model2services, model2descriptions, model2ontologies, and model2semantics. Table 1 shows the rules that are considered for automatically translating a UML class diagram into RESTful services. For any UML class that is annotated with the ≪SOAP≫ stereotype, an equivalent Java-based SOAP service, annotated with the @WebService keyword, is generated. In addition, for all the operations of the ≪SOAP≫ annotated UML class, equivalent Java methods would be annotated with the @WebMethod as prescribed by the JAX-WS specification. These rules are then implemented with the aid of the Acceleo framework. Listing 1 illustrates the snippet of the Acceleo template implementing the mappings in Table 1.

These templates are text-based and can be effortlessly defined for transformation rules to any high-level programming language and description languages meant for SWS.

7.2 Semantics layer

The Semantics Description Language Selector and the Semantics Descriptor modules depend on the Model2Code Transformer module, and were implemented via desperate transformation rules such as model2semantics. These rules comprise a number of

```
<%if (hasStereotype("SOAP")){%>
package <%package.name%>
import javax.jws.WebMethod;
import javax.jws.WebService;

/**
* @WebService
* Important for decorating our class as a SOAP Web Service
*/
 @WebService(serviceName = "<%name%>",
                                        portName = "<%name%>Port",
                                        endpointInterface = "<%package.name%>.<%name%>Interface",
                                        targetNamespace = "http://<%package.name.reverse()%>",
                                        wsdlLocation="
WebContent/wsdl/<%name.toLowerCase()%>.wsdl")
<%visibility%> class <%name%> {
…
```

Listing 1: Acceleo template for UML2SOAP mappings

UML Class Diagram	OWL-S Profile
<<OWL-S>><<profile>> Stereotype	OWL-S Profile
Class name	Class, Service Name, Profile Name
Operation parameters (in) (return)	&process (#input, #output, #parameter , #results)
Generalization	SubClassOf
Constraints	&expr (#condition), restrictions, cardinality
Enumerations, EnumerationLiterals	Collections, OneOf

Table 2: UML2OWL-S mappings

mappings, including UML2WSMO and UML2OWL-S. The selector mainly infers the semantic description language of choice, based on the service model's annotations. This simply means that the developer annotates the service model using specific stereotypes, such as ≪OWL-S≫ and the required code is then automatically generated using the Semantics Descriptor. The selector does not restrict the number of languages that could be selected simultaneously.

Table 2 highlights the UML2OWL-S transforming rules that are responsible for translating an annotated service model with ≪OWL-S≫≪profile≫ stereotypes into various OWL-S service profiles.

Every OWL-S annotated class name is mapped to an OWL-S class, Service Name, and Profile Name according to the OWL-S specifications [18]. The UML class operation input and output parameters are mapped to OWL-S Profile properties such as Inputs, Output, Parameters, and Results. UML-defined constraints are then aligned to OWL-S logical expressions in Preconditions and Effects.

7.3 Knowledge layer

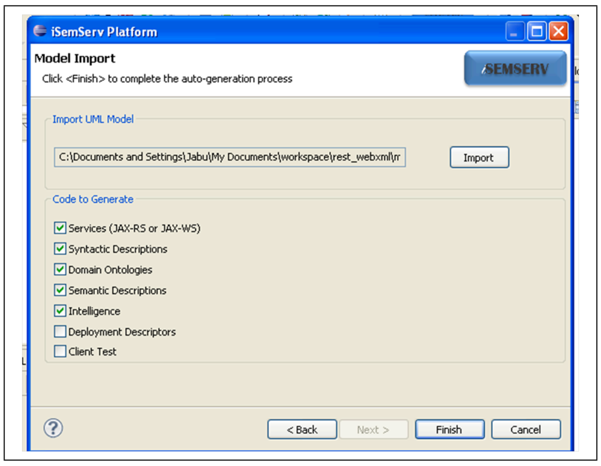
In this layer, we implemented the Ontology language Selector and Domain Ontologies Generator using the iSemServ UML Profile and the Acceleo defined transformation rules and templates. For instance, for every class name annotated with the ≪WSMO≫ and ≪ontology≫ stereotypes, UML class operations and enumerations are automatically translated into corresponding WSML ontological concepts by the Domain Ontologies Generator. The properties of the UML class and the input parameters of operations are then translated to the attributes of the relevant WSMO concepts. For example, in Listing 2 the concept "person" would have been generated from a UML class with the name "person". The attributes firstName and lastName would have been the properties of the same class.

Although the iSemServ approach is structured

```
concept person
nonFunctionalProperties
                    dc#description hasValue "WSML description of a person"
endNonFunctionalProperties
   firstName ofType _string
   lastName ofType _string
```

Listing 2: WMSO concept and attributes

into various layers with disparate modules, the front-end system representing the implemented platform is only an Eclipse plug-in that could easily be integrated within version 3.5 of Eclipse Galileo. The user interface that the developer interacts with is depicted in Figure 4.



Model Import

Click <Finish> to complete the auto-generation process

Import UML Model

C:\Documents and Settings\Jabu\My Documents\workspace\rest_webxml\m

Code to Generate

- [✓] Services (JAX-RS or JAX-WS)
- [✓] Syntactic Descriptions
- [✓] Domain Ontologies
- [✓] Semantic Descriptions
- [✓] Intelligence
- [ ] Deployment Descriptors
- [ ] Client Test

Figure 4: iSemServe Eclipse plug-in

The service developer only needs to import a UML compliant packaged service model, and select the types of service elements that need to be auto-generated. The plug-in would then, in the background, semi-automatically generate all the selected service elements depending on the model annotations. The developer could then review, edit, and finalize the generated modules or even reuse previously generated elements (e.g. domain ontology) using various editors integrated within the Eclipse environment as noted in the discussion of the proposed solution.

8 EVALUATION OF ISEMSERV

In order to ensure validity and utility of the proposed model-driven iSemServ approach and its implementation, the evaluation process plays an important part. Moreover, evaluation is essential to the development of any technical solution. For the proposed solution, different types of evaluations were conducted using practical use-case scenarios and comparative analysis in order to qualitatively note the benefits and the limitations of our solution against other related solutions.

8.1 Comparative analysis

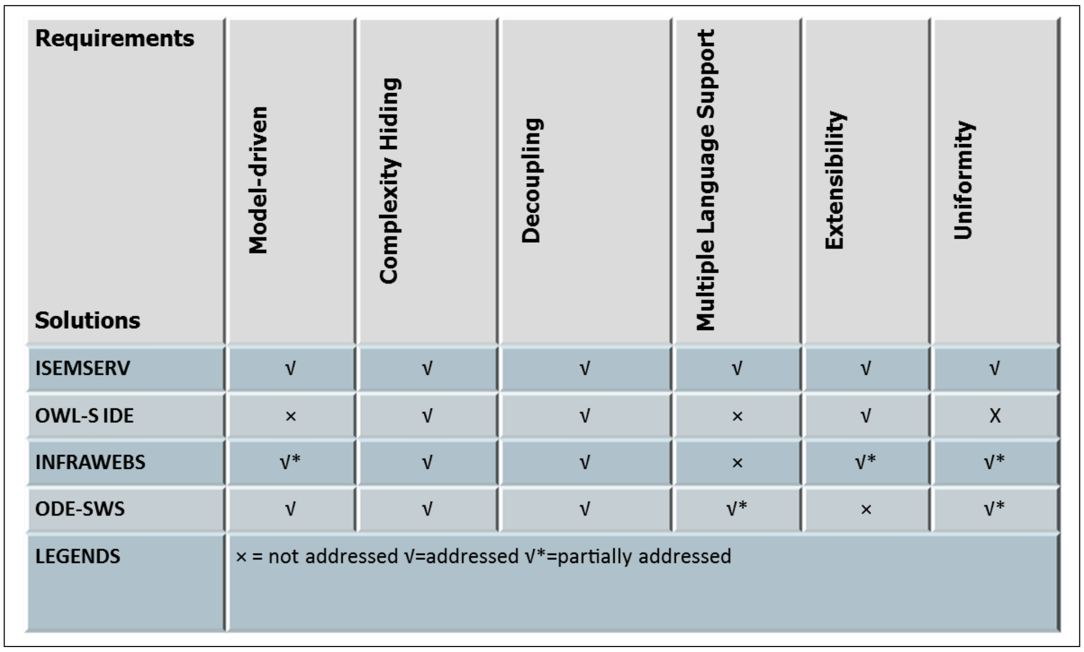
Comparative analysis [41] plays an important role in assessing any new solution against the existing similar solutions. The existing solutions that formed part of the analysis were discussed under the related work section. The comparative criteria used for evaluation is based on the design requirements presented in section 5.

As may be noted in Table 3, our proposed iSemServ model-driven approach fills the gap that currently exists in the literature by providing an environment that makes it possible to develop SWS using semantic and syntactic description languages of choice. ODE-SWS also supports multiple description languages, but only for semantic descriptions.

The majority of the solutions that were evaluated pay attention to the principle of complexity hiding when it comes to simplifying the process of building SWS. Our solution addresses the issue of complexity hiding through the auto-generation of the code necessary for all SWS elements (i.e. service, descriptions, and ontologies). However, it should be noted that the developer still needs to understand and be proficient in the description languages of choice in order to augment or edit the generated skeletons of code.

From the evaluations, it was found that only our proposed solution fully facilitates the development of SWS within a unified environment. The unified environment is demonstrated in Figure 3. INFRAWEBS partially addresses the design requirement of a unified development environment by allowing the import of existing syntactic services and the creation of semantic descriptions for imported services. However, in INFRAWEBS, the process of building syntactic services anew is not considered.

The solutions that were found to be supporting the extensibility design requirement were iSemServ and OWL-IDE. Both of these solutions adopt the plug-in principle found in development environments such as Eclipse. As may be noted from Table 3, all the evaluated solutions adhere to the decoupling design requirement.



Requirements / Solutions	Model-driven	Complexity Hiding	Decoupling	Multiple Language Support	Extensibility	Uniformity
ISEMSERV	√	√	√	√	√	√
OWL-S IDE	×	√	√	×	√	X
INFRAWEBS	√*	√	√	×	√*	√*
ODE-SWS	√	√	√	√*	×	√*
LEGENDS	× = not addressed √=addressed √*=partially addressed					

Table 3: iSemServ Comparative Evaluation

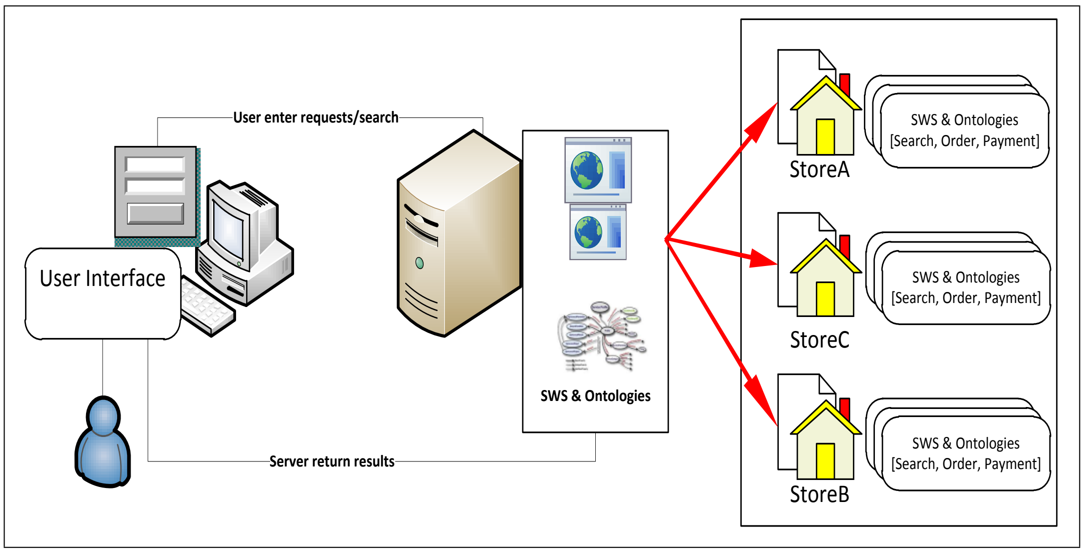

User enter requests/search
User Interface
SWS & Ontologies
Server return results
StoreA
SWS & Ontologies [Search, Order, Payment]
StoreC
SWS & Ontologies [Search, Order, Payment]
StoreB
SWS & Ontologies [Search, Order, Payment]

Figure 5: Online multimedia trading use case scenario

8.2 Practical use case scenario

The scenario depicted in Figure 5 demonstrates how Semantic Web Services could be designed and semi-automatically developed in a simple and efficient manner by using the iSemServ plug-in. It should be noted that the main purpose of the practical use case scenario was to demonstrate the underlying benefits of the proposed solution particularly in addressing the articulated challenges, and understanding some of the limitations that are inherent to iSemServ.

This real-world scenario involves tasks that have been assigned to the service developer. The tasks involve developing an online multimedia trading Web application that enables service providers to automate a number of activities involved when customers buy multimedia items (e.g., CDs and DVDs) from different online stores. Some of the repeatable processes that the sellers would like to automate include:

1. search for different products in a semantically enabled multimedia catalogue,
2. order products from the shopping cart,
3. use external services to make payments,

among others.

It should also be noted that our goal in this section is not to practically demonstrate the complete functionality of the online multimedia trading scenario or the value of SWS, but the main goal is to demonstrate how the proposed solution could be used by a developer to realize some of the disparate artefacts that make up the scenario.

From the proposed solution perspective, such a scenario could be implemented using different languages. Furthermore, since there are a number of sellers involved, each seller could opt to implement their SWS

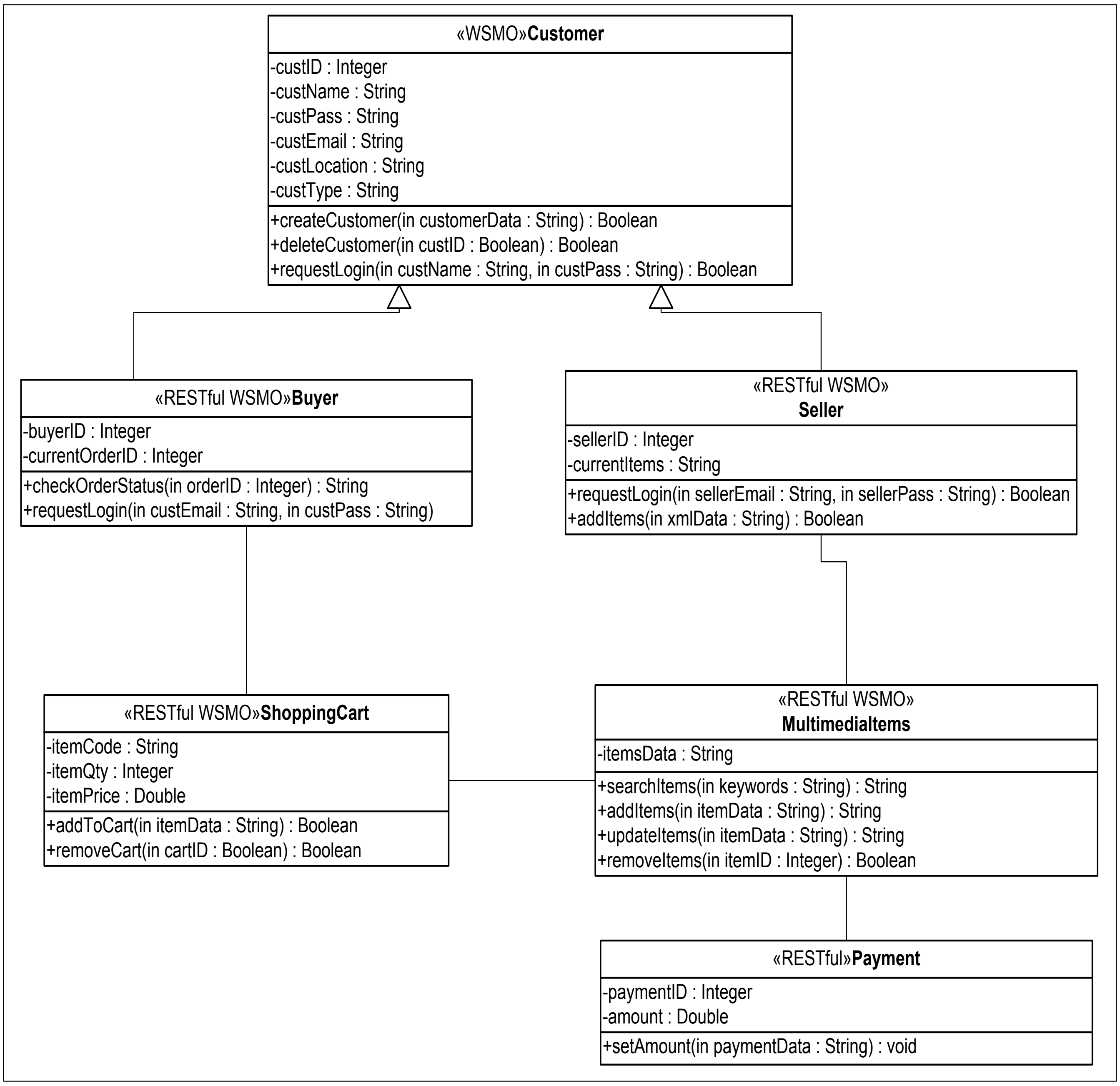

«WSMO»Customer
-custID : Integer
-custName : String
-custPass : String
-custEmail : String
-custLocation : String
-custType : String
+createCustomer(in customerData : String) : Boolean
+deleteCustomer(in custID : Boolean) : Boolean
+requestLogin(in custName : String, in custPass : String) : Boolean
«RESTful WSMO»Buyer
-buyerID : Integer
-currentOrderID : Integer
+checkOrderStatus(in orderID : Integer) : String
+requestLogin(in custEmail : String, in custPass : String)
«RESTful WSMO» Seller
-sellerID : Integer
-currentItems : String
+requestLogin(in sellerEmail : String, in sellerPass : String) : Boolean
+addItems(in xmlData : String) : Boolean
«RESTful WSMO»ShoppingCart
-itemCode : String
-itemQty : Integer
-itemPrice : Double
+addToCart(in itemData : String) : Boolean
+removeCart(in cartID : Boolean) : Boolean
«RESTful WSMO» MultimediaItems
-itemsData : String
+searchItems(in keywords : String) : String
+addItems(in itemData : String) : String
+updateItems(in itemData : String) : String
+removeItems(in itemID : Integer) : Boolean
«RESTful»Payment
-paymentID : Integer
-amount : Double
+setAmount(in paymentData : String) : void

Figure 6: Use case scenario service model

in the languages of choice. However, because the consumer does not care much about the underlying implementation or languages used to implement the services, what is important is that the system is implemented in a manner that allows for interoperability in terms of different ontologies and semantic descriptions.

For experimentation purposes, we therefore opted to demonstrate the relevance of our solution by implementing most of the services using the RESTful architectural style and WSMO semantic model. However, in order to also demonstrate the support for multiple languages, some elements were auto-generated using OWL-S. Furthermore, in demonstrating the ability of our solution to interoperate with already existing elements, available domain ontologies describing multimedia products[4] were also exploited.

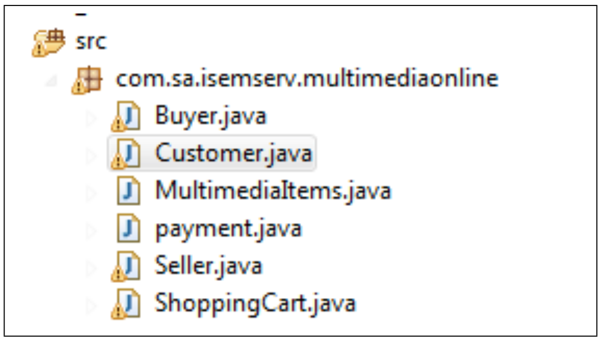
In developing some elements of the highlighted scenario, the following steps were taken using the iSemServ plugin as depicted in Figure 4. The service developer used the UML2 SDK plugged into Eclipse to design service models capturing both the services' structures and semantic concepts. The services model is defined according to the iSemServ UML profile. The partial service model in the form of a UML class diagram is illustrated in Figure 6.

As may be noted, 6 classes are modelled and annotated with appropriate keywords (e.g. ≪RESTful≫ and ≪WSMO≫). From the service model, syntactic RESTful services are generated according to the ≪RESTful≫ annotation. In this regard, the iSemServ environment facilitates the generation of skeleton syntactic RESTful services. The amount of time it

[4]`http://www.wsmo.org/ontologies/amazonECS/amazonOntology.wsml`.

takes, for example, to generate the skeleton code for the classes depicted in the model is only a few milliseconds compared with manually coding the structure of RESTful services. However, this is not novel, as this method is used extensively in a number of mature development environments, such as Eclipse and Visual Studio. The key difference is that in the iSemServ platform, the service developer is in control of what code skeletons could be generated through the use of service models and profiles.

A snippet of the generated code structure is shown in Figure 7. This structure demonstrates the number of Java classes, representing RESTful services, generated based on the number of classes modelled in UML.

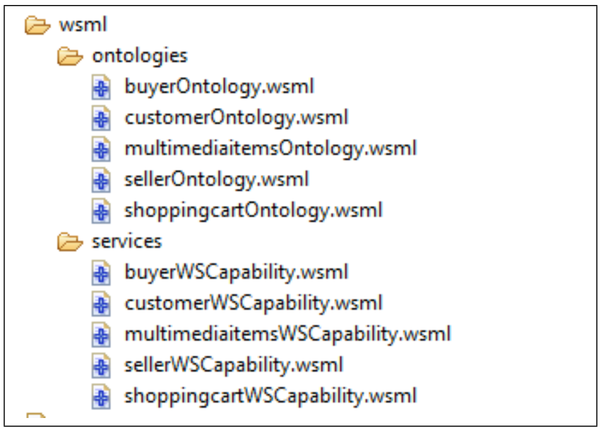


```
src
  com.sa.isemserv.multimediaonline
    Buyer.java
    Customer.java
    MultimediaItems.java
    payment.java
    Seller.java
    ShoppingCart.java
```

Figure 7: Syntactic RESTful services

As illustrated in the service model (cf. Figure 6), the five classes in UML represent five RESTful services, while one, viz. the Customer class, is not a RESTful service, but a pure Java POJO class. Nevertheless, semantic descriptions and domain ontologies for this class are also generated on the basis of the ≪WSMO≫ annotation.

Semantic descriptions and domain ontologies are dynamic, in a sense that they evolve over time. As a result, they are resource intensive to build and update. In Figure 8, the snippet of semantic descriptions and domain ontologies auto-generated for each class annotated with ≪WSMO≫ stereotype are shown.



```
wsml
  ontologies
    buyerOntology.wsml
    customerOntology.wsml
    multimediaitemsOntology.wsml
    sellerOntology.wsml
    shoppingcartOntology.wsml
  services
    buyerWSCapability.wsml
    customerWSCapability.wsml
    multimediaitemsWSCapability.wsml
    sellerWSCapability.wsml
    shoppingcartWSCapability.wsml
```

Figure 8: Generated semantic descriptions

Basically, the [ClassName]+WSCapability.wsml files represent Web Service capabilities according to WSMO specifications. The generated WS capability skeleton code for the CheckOrderStatus and RequestLogin operations is demonstrated in Listing 3. The service engineer could further edit the generated code using semantic tools, such as the WSMO editor embedded within the Eclipse environment. The code is generated based on the mapping rules discussed in Section 7.

As may be noted in Listing 3, Lines 10-25, illustrate the auto-generated semantic descriptions (e.g. service name, capability name, and non-functional properties) necessary for semantically enabling the buyer RESTful service. Nevertheless, the developer could change the descriptions to suit own development requirements. In addition, Line 11 indicates that external ontologies can also be referenced for purposes of augmenting the generated descriptions. The rest of the code is directly linked to the necessary mappings (e.g., UML2WSMO).

For purposes of demonstrating that the solution supports diverse service and semantic description languages, Figure 9 shows an example of a partial OWL-S service grounding that was auto-generated by the proposed solution when the multimedia items class was annotated with the ≪OWL-S≫ stereotype. This service grounding shows the auto-generated operations that could be invoked by an agent within the Multimedia Items RESTful service. It should be noted that all of these elements are auto-generated based on the service model as depicted in Figure 6.

The evaluation approach of using the proposed solution to partially implement the use case scenario has provided several insights with regard to the design and development of semantic services. The following benefits were observed:

1. The development of different building blocks that make up semantic services can be realized within a unified environment
2. The development effort is reduced through the auto-generation of different implementation artefacts, which may lead to high development times and costs, if a manual approach is followed
3. The service developer controls the development life cycle, and the iSemServ plug-in does not impose restrictions on the types of service or the semantic descriptions languages. The only requirement pertains to the usage of UML-based service models for structuring services and domain knowledge
4. The implemented Eclipse plugin interoperated effectively with other tools in Eclipse (e.g. UML2SDK), demonstrating that our solution could exploit other tools to simplify the generation of domain ontologies and semantic descriptions.

The iSemServ solution presented in this article has some practical limitations as well. For instance, key features such as dynamic semantic services discovery, selection, composition, and monitoring are not addressed. Furthermore, checking the correctness and validity of the auto-generated skeleton code is not supported by the proposed solution. This task is left to the developer, which can be a tedious process when dealing with a large set of code and descriptions. Nevertheless,

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
comment <!--Generated by SemServ Model2Semantics transformer using Acceleo 2.8-->
comment <!--Date: November 25, 2013 [11:19:42 AM] -->
namespace { _"http://www.isemserv.co.za/services/buyerSemantics#",
buy _"http://www.isemserv.co.za/ontologies#",
dc _"http://purl.org/dc/elements/1.1#",
wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
xsd  _"http://www.w3.org/2001/XMLSchema#",
desc _"http://www.isemserv.co.za/descriptions#"}

   webService checkOrderStatusrequestLoginService

   importsOntology {_"http://www.wsmo.org/ontologies/amazonECS/amazonOntology.wsml"}
   capability checkOrderStatusrequestLoginCapability

   nonFunctionalProperties
   dc#typehasValue"service ontology"
   dc#descriptionhasValue"Enter description for this capability"
   dc#titlehasValue"Capability for a buyer Web service"
   dc#creatorhasValue {"Your Name"}
   dc#publisherhasValue"isemserv"
   dc#datehasValue"Nov 25, 2013 [11:19:42 AM]"
   dc#typehasValue _"http://www.wsmo.org/2004/d2#ontologies"
   dc#identifierhasValue _"http://www.isemserv.co.za/services/buyer"
   dc#languagehasValue"en-US"
   dc#formathasValue"text/plain"
   desc#serviceDescription hasValue"COMPLETE URL FOR SERVICE DESCRIPTION"
   endNonFunctionalProperties

   sharedVariables {?orderID}
   sharedVariables {?custEmail, ?custPass}

   precondition
   nonFunctionalProperties
   dc#descriptionhasValue"condition(s) that need to be satisfied before service is invoked"
   endNonFunctionalProperties
   definedBy
   ?orderID memberOf OrderID
   ?custEmail memberOf CustEmail
   ?custPass memberOf CustPass
```

Listing 3: Partial WMSO service capability

one of the key principles of our solution is extensibility, which is intended to enable other researchers and developers to extend the iSemServ solutions with any required modules via the Eclipse environment.

9 CONCLUSION

Semantic Web Services will play a significant role in revolutionizing the current Web, where business processes would be executed and automated by machines with minimal user interventions. However, implementation support for such services is lacking in real-life environments, due to a number of challenges such as tedious and error-prone SWS development processes, lack of integration of SWS solutions with other emerging Web service technologies, and the complexities of semantic description languages.

Although there are a number of relevant and successful case studies in the SWS space with regard to solutions that purports to address some of these issues, most of them do not appreciate the diversity of the different semantic and syntactic description languages that exist and may even emerge. Most of the existing tools are tightly coupled to one specific language and ignoring others, thus leading to restrictive and exclusive SWS development environments.

The main objective of the article was to devise a model-driven approach that would facilitate and semi-automate the construction of SWS using a variety of description languages. This approach was produced by following the design science research (DSR) methodology, which is a problem-solving technique that leads to the development of novel IT artefacts.

The proposed model-driven approach, termed iSemServ, was designed following the decoupling, integration, extensibility, multiple language support, extensibility, and complexity hiding design requirements, which were derived by evaluating existing solutions, related literature, and the identified challenges experienced when building SWS. The iSemServ approach is made up of three modular layers that function independently from each other. These layers consist of all the important components that adhere to the design requirements, such as the service modeller, semantic descriptions language selector, and domain ontologies generator.

The proposed approach was implemented using open technologies embedded in the widely adopted

```
<grounding:WsdlGrounding rdf:ID="MultimediaOnlineGrounding">
<service:supportedBy
rdf:resource="http://www.isemserv.co.za/services/MultimediaOnlineProcess.owl"/>
<grounding:hasAtomicProcessGrounding rdf:resource="#SearchItems"/>
<grounding:hasAtomicProcessGrounding rdf:resource="#AddItems"/>
<grounding:hasAtomicProcessGrounding rdf:resource="#UpdateItems"/>
<grounding:hasAtomicProcessGrounding rdf:resource="#RemoveItems"/>
</grounding:WsdlGrounding>
```

Figure 9: Partial OWL-S multimedia items service grounding

Eclipse framework. Some of the contributions in the proposed approach include the iSemServ UML profile that enables the use of multiple description languages through flexible model annotations. Another important contribution is the definition of the mapping rules and templates that are necessary for automatically generating a number of elements that make up the SWS from a service model created using MDA-compliant standards, such as UML.

The iSemServ solution was tested for validity and relevance through a practical use case scenario and by performing a comparative analysis against the identified similar solutions. The results indicate that the proposed solution addresses the challenge of supporting multiple description languages for building SWS. It further supports uniformity and extensibility requirements, which are important in ensuring that existing and future semantic and syntactic description languages could be accommodated. Overall, the proposed iSemServ solution succeeds in addressing one of the pertinent issues, restrictive and exclusive environments, faced by developers when developing SWS. The proposed solution is presented as simple and yet useful for supporting average and expert service engineers in building SWS using syntactic and semantic descriptions of choice.

Further research of this work points to the advancements of the transformation rules, code generation templates, and the UML profile to enable the autogeneration of SWS elements that are more complete and expressive.

REFERENCES

[1] D. Bachlechner and K. Fink. "Semantic Web Service research: Current challenges and proximate achievements". *IJCSA*, vol. 5, no. 3b, pp. 117–140, 2008.

[2] J. d. Bruijn, D. Fensel, U. Keller and R. Lara. "Using the web service modeling ontology to enable semantic e-business". *Communications of the ACM*, vol. 48, no. 12, pp. 43–47, 2005.

[3] V. Janev and S. Vraneš. "Applicability assessment of Semantic Web technologies". *Information Processing & Management*, vol. 47, no. 4, pp. 507–517, 2011.

[4] B. Blake, L. Cabral, B. König-Ries, U. Küster and D. Martin. *Semantic Web Services: Advancement through evaluation.* Springer, 2012.

[5] T. Wahl and G. Sindre. "A survey of development methods for semantic web service systems". In *Information systems and new applications in the service sector: Models and methods*, pp. 117–132. IGI Global, 2011.

[6] R. Witte, B. Sateli, N. Khamis and J. Rilling. "Intelligent software development environments: integrating natural language processing with the eclipse platform". In *Advances in Artificial Intelligence*, pp. 408–419. Springer, 2011.

[7] G. Agre, Z. Marinova, T. Pariente and A. Micsik. "Towards Semantic Web Service engineering". *Service Matchmaking and Resource Retrieval in the Semantic Web*, p. 91, 2007.

[8] J. Cardoso. *Semantic Web services: Theory, tools, and applications.* IGI Global, 2007.

[9] J. Mtsweni, E. Biermann and L. Pretorius. "iSemServ: Towards the engineering of intelligent semantic-based services". In *Current Trends in Web Engineering*, pp. 550–559. Springer, 2010.

[10] H. Barros, A. Silva, E. Costa, I. I. Bittencourt, O. Holanda and L. Sales. "Steps, techniques, and technologies for the development of intelligent applications based on Semantic Web Services: A case study in e-learning systems". *Engineering Applications of Artificial Intelligence*, vol. 24, no. 8, pp. 1355–1367, 2011.

[11] J. Shen, G. Beydoun and G. Low. "A multi agent system based implementation for semantic web services". *Information systems development: Reflections, challenges and new directions*, pp. 231–241, 2013.

[12] A. Hevner, S. March, J. Park and S. Ram. "Design science in information systems research". *MIS Quarterly*, vol. 28, pp. 75–105, 2004.

[13] T. Berners-Lee, J. Hendler, O. Lassila et al. "The semantic web". *Scientific american*, vol. 284, no. 5, pp. 28–37, 2001.

[14] J. Lu, D. Ruan and G. Zhang. *E-Service intelligence: Methodologies, technologies and applications.* Studies in computational intelligence. Springer, 2007.

[15] D. Amar Bensaber and M. Malki. "Development of semantic web services: Model driven approach". In *Proceedings of the 8th international conference on New technologies in distributed systems*, p. 40. ACM, 2008.

[16] R. Chinnici, J.-J. Moreau, A. Ryman and S. Weerawarana. "Web services description language (WSDL) version 2.0 part 1: Core language". *W3C recommendation*, vol. 26, p. 19, 2007.

[17] M. J. Hadley. "Web application description language (WADL)". *W3C Member Submission*, 2006.

[18] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne et al. "OWL-S: Semantic markup

for web services". *W3C member submission*, vol. 22, pp. 2007–04, 2004.

[19] D. Roman, J. De Bruijn, A. Mocan, H. Lausen, J. Domingue, C. Bussler and D. Fensel. "WWW: WSMO, WSML, and WSMX in a nutshell". In *The Semantic Web–ASWC 2006*, pp. 516–522. Springer, 2006.

[20] J. Nern, G. Agre, T. Atanasova, Z. Marinova, A. Micsik, L. Kovács, J. Saarela and T. Westkaemper. "INFRAWEBS semantic Web service development on the base of knowledge management layer". *International journal on information theories and application (IJITA)*, 2006.

[21] J. Mtsweni. "Exploiting UML and acceleo for developing semantic web services". In *International Conference for Internet Technology and Secured Transactions, London, UK, 2012*, pp. 753–758. 2012.

[22] D. Fensel, F. Fischer, J. Kopeckỳ, R. Krummenacher, D. Lambert and T. Vitvar. "WSMO-Lite: Lightweight semantic descriptions for services on the web". *W3C Member Submission*, vol. 23, 2010.

[23] T. R. Gruber. "A translation approach to portable ontology specifications". *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[24] R. Studer, S. Grimm and A. Abecker. *Semantic web services: concepts, technologies, and applications.* Springer, 2007.

[25] I.-W. Kim and K.-H. Lee. "A model-driven approach for describing semantic web services: From UML to OWL-S". *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 39, no. 6, pp. 637–646, 2009.

[26] D. Kuropka, P. Trger, S. Staab, M. Weske, D. Kuropka, P. Trger, S. Staab and M. Weske. *Semantic service provisioning.* Springer Publishing Company, Incorporated, 2008.

[27] O. F. Ferreira Filho and M. A. G. V. Ferreira. "Semantic web services: A RESTful approach". In *Proceedings of the IADIS International Conference on WWW/Internet*, pp. 169–180. 2009.

[28] P. M. Kelly, P. D. Coddington and A. L. Wendelborn. "A simplified approach to web service development". In *Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54*, pp. 79–88. Australian Computer Society, Inc., 2006.

[29] O. Corcho, A. Gómez-Pérez, M. Fernández-López and M. Lama. "ODE-SWS: A semantic web service development environment". In *Proceedings of the 2003 1st international workshop on semantic web and databases (SWDB03).* Universidad de Berlin, 2003.

[30] WebODE. "WebODE ontology engineering platform". *Available: http://webode.dia.fi.upm.ex/WebODEWeb/index.html*, April 2003.

[31] N. Srinivasan, M. Paolucci and K. Sycara. "Semantic web service discovery in the OWL-S IDE". In *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, vol. 6, pp. 109b–109b. IEEE, 2006.

[32] V. Vaishnavi and W. Kuechler. "Design science research in information systems". *Available: http://www.desrist.org/design-research-in-information-systems/*, 2004.

[33] S. Weber, R. Beck and R. Gregory. "Combining design science and design research perspectives: Findings of three prototyping projects". In *45th Hawaii International Conference on System Science (HICSS)*, pp. 4092–4101. 2012.

[34] M. S. Olivier. *Information technology research: A practical guide for computer science and informatics.* Van Schaik, 2009.

[35] C. Pahl. "Semantic model-driven architecting of service-based software systems". *Information and software Technology*, vol. 49, no. 8, pp. 838–850, 2007.

[36] M. Nassar, A. Anwar, S. Ebersold, B. Elasri, B. Coulette and A. Kriouile. "Code generation in VUML profile: A model-driven approach". In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, pp. 412–419. IEEE, 2009.

[37] W. Sun, S. Li, D. Zhang and Y. Yan. "A model-driven reverse engineering approach for semantic web services composition". In *Software Engineering, 2009. WCSE'09. WRI World Congress on*, vol. 3, pp. 101–105. IEEE, 2009.

[38] F. Lautenbacher. "A UML profile and transformation rules for semantic web services". Tech. rep., University of Augsburg, Germany, 2006.

[39] R. Akkiraju, J. Farrell, J. A. Miller, M. Nagarajan, A. Sheth and K. Verma. "Web service semantics (WSDL-S)". *W3C Member Submission*, 2005.

[40] "Acceleo—transforming models into code", April 2011. URL `http://www.eclipse.org/acceleo`.

[41] E. Hofstee. *Constructing a good dissertation: A practical guide to finishing a master's, MBA or PhD on schedule.* EPE, 2006.