A MODEL-DRIVEN APPROACH TO CLOUD SAAS INTEROPERABILITY
5. A MODEL-DRIVEN APPROACH TO CLOUD SAAS INTEROPERABILITY

5.1. Introduction

A cloud is an amalgam of diverse technologies and platforms. The software applications in the cloud may be implemented using varied technologies and deployed in different computing environments. These applications residing in the cloud may not exist in isolation. Irrespective of the technologies used to implement them, they may require communicating with each other and exchange information transparently. Thus, interoperability among the cloud SaaS is a relevant and significant issue in Cloud computing. Also, as mentioned earlier, interoperability is one of the primary goals of MDA, besides reusability and portability.

In general, the interactions among the heterogeneous applications take place through well-defined interfaces. For an effective and efficient communication, these formal interfaces must be independent of the technology platforms on which the applications are implemented. The application interface itself may be implemented using specific technologies. As the technologies evolve, the interface definition may become obsolete and need to be replaced. In the given scenario, instead of directly specifying the interface using available technology, the MDA approach may be leveraged to specify platform independent model of the interface, which may then be targeted on several specific implementation technologies.
A model-driven approach to ensure interoperability among the cloud SaaS is the primary objective of our research work and is the focus of this chapter. Section 5.2 briefly presents an overview of the interoperability issue among the software applications in general and cloud SaaS in particular. Section 5.3 discusses incorporation of MDA approach in development of cloud software service interface. A PIM and a WSDL PSM of the cloud service interface is specified in this section. The elements of Web Service Description Language (WSDL 2.0) and the various message exchange patterns (MEP) are discussed in section 5.4. This section also highlights the structural differences between WSDL 1.1 and WSDL 2.0. Since a model must conform to its metamodel a WSDL Metamodel is presented in section 5.5. The transformation definition specifying the mappings for PIM to WSDL PSM transformation are briefly discussed in Section 5.6. Section 5.7 models interoperability using UML class diagram. Section 5.8 illustrates the formalism of the transformation definition on a specific technology platform, as implemented by the authors. The conclusion of the chapter is presented in Section 5.9.

5.2. Addressing Interoperability in a Cloud SaaS

Interoperability among the software components plays a significant role in distributed computing where exchange of messages becomes vital for achieving a common goal.

In early- and mid-90s, the sharing of data among the applications on same computer or different computers was facilitated through the then prevalent technologies such as Microsoft’s COM/DCOM (Component Object Model/Distributed Component Object Model) and OMG’s CORBA/IIOP (Common Object Request Broker Architecture/Internet Inter-ORB Protocol), which allowed the units of functionality to be reused as binary objects. Sun created its own native Object Request Broker (ORB), called RMI (Remote Invocation Method). But these technologies were platform-specific.
For example, COM/DCOM was tied to Windows platform and though CORBA/IIOP did not have this limitation, it was used with non-Java languages. Also, in the absence of a universal standard for data representation at that time, the communication among the applications was carried out in binary form; but this binary data was restricted by firewalls.

The year 1999-2001 witnessed the advent of Web services. Web-services are characterized by non-proprietary standards, cross-platform capabilities and human-readable messages that can be sent across firewalls. Since the web service message is in human-readable form, it requires more bytes to transmit the same amount of information as compared to that of its predecessors.

Web Services offer a suitable technology platform for realizing SOA. The core components of the Web services framework include – web services, service descriptions and messages. A service description document describing the functionality offered by a service via its interface, accompanies each service. This document is specified in WSDL. WSDL is not tied to any specific XML messaging system such as XML-RPC or SOAP, but it does include built-in extensions for describing SOAP services. Once a WSDL document (.wsdl file) is specified for a service, a SOAP client can be manually created to invoke any of the service’s publicly available functions as defined in the WSDL document. Alternatively, the service can be automatically invoked using a WSDL invocation tool available for the purpose such as Web Service Invocation Framework (WSIF) from IBM, SOAP::Lite, GLUE from The Mind Electric etc. These tools may be command-line based or may provide a web-based interface [57].

Since a cloud is a metaphor for the Internet, the Web services technology may be ideally used for defining the cloud software services (Cloud SaaS). Thus, the standard-based
service descriptions would be the key ingredients in establishing a consistently loosely coupled form of communication among the services implemented in the cloud.

5.3. Incorporating MDA Approach in the Development of Cloud Software Services

An imperative issue related to technology evolution is platform volatility, that is, the tendency for the platform ground to shift under one's feet [22]. In straight terms, it refers to the obsolescence of platform technology. Although platform volatility is mentioned in relation to enterprise technology, but it also surfaces in the sphere of information exchange because of the proliferation of communication and messaging frameworks. In this regard, an MDA-based approach to the development of cloud services would play a significant role in mitigating the undesirable effects of technology change. A modeling language such as UML may be used to model different aspects of cloud services; and transformation rules may be defined for performing model transformations. Besides, MDA may be leveraged to compose new cloud services from existing services as proposed in [78] [79].

This section illustrates an MDA-based modeling of the OHRS cloud SaaS (application) as a service, namely the OHRS_Service. This service may be availed by any small or medium scale hotel enterprise. The OHRS_Service provides various online business functionalities to a hotel enterprise and hotel customers such as book an accommodation, cancel a booking, check availability of accommodation, and generate reports for the management etc.

Once again, UML has been used to specify the PIM and PSM of the OHRS_Service. The PIM of the OHRS cloud service provides a formal definition of the operations offered by
A Model-Driven Approach to Cloud SaaS Interoperability

<< Cloud Software Service >>

**OHRS Service**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bookAccommodation</code></td>
<td>Accepts customer information and books an accommodation.</td>
</tr>
<tr>
<td><code>cancelAccommodation</code></td>
<td>Cancels an existing accommodation.</td>
</tr>
<tr>
<td><code>checkAvailability</code></td>
<td>Checks availability of a specific unit type for a given period.</td>
</tr>
</tbody>
</table>

**context OHRS_Service :: bookAccommodation**

- **pre:**
  - The number of units booked must be less than or equal to the number of units available:
    \[ \text{units_booked} \leq \text{available_units} \]
  - The date of booking must be less than or equal to the current date:
    \[ \text{dt_of_booking} \leq \text{current_date} \]
  - The `book_from` date must be less than or equal to the `book_to` date:
    \[ \text{book_from} \leq \text{book_to} \]

**post:**
- The number of available units is reduced by the number of units booked:
  \[ \text{available_units} = \text{available_units} - \text{units_booked} \]

**context OHRS_Service :: cancelAccommodation**

- **pre:**
  - The `custID` must exist in the Customer table of Hotel database:
    \[ \text{request.custID} = \text{Hoteldb.Customer.custID} \]

**context OHRS_Service :: checkAvailability**

- **pre:**
  - The requested unit type must exist in the Hotel database:
    \[ \text{request.unit_type} = \text{Hoteldb.unit_type} \]
  - The number of units requested must be less than or equal to the number of available units:
    \[ \text{units_requested} \leq \text{available_units} \]
  - The `book_from` date must be less than or equal to the `book_to` date:
    \[ \text{book_from} \leq \text{book_to} \]

---

Figure 5.1 A Formal Model (PIM) of a Business Service
the service that can be accessed through the OHRS_Service interface. Although, the service offers a range of operations through its interface, for the sake of simplicity, only a couple of operations associated with the service are depicted in Figure 5.1. The business rules for the operations are specified by declaring the constraints as pre-conditions, post-conditions and invariants in Object Constraint Language (OCL) and are depicted in the lower part of the Figure 5.1 [80]. Combining UML with OCL results in a PIM specification that is more precise, rigorous and semantically rich. The PIM is then transformed into a PSM targeted on WSDL [22][81][82].

Figure 5.2 depicts the mappings of the source model (PIM) constructs to the target model (PSM) constructs. Again, to keep the diagram simple, mapping for only one operation is exhibited. The 'checkAvailability' operation in service PIM and PSM takes the type of unit, the number of units and the dates specifying the booking period as input, and provides the availability status as output. As depicted, in Figure 5.2, the interface of the OHRS_Service PIM is mapped to the interface element in WSDL PSM. An interface of a service exposes a number of functionalities to its client. The functionality corresponds to an operation or a method. In this figure, the 'checkAvailability' operation in the PIM is mapped to the corresponding operation element of WSDL PSM. An operation may have some input and output parameters (messages) associated with it, which are used to pass data values to the operation or to return values from it. The input and output parameters in the service PIM are mapped to the input and output elements respectively, in the WSDL PSM.
As discussed earlier, an essential requirement of the MDA approach is that the models defined at different levels of abstraction during the process of development of the software system must conform to their respective metamodels. The platform-specific WSDL model must therefore be in conformance with the WSDL metamodel. A metamodel for WSDL 1.1 is discussed in [83][84][85]. A metamodel for WSDL 2.0 has been described by the authors in Section 5.5.

In the context of MDA, the WSDL represents the PSM of the cloud SaaS interface. The WSDL in a cloud SaaS is responsible for message exchange and communication among the interacting services, thus ensuring interoperability at the PSM level in the model-driven approach.

5.4. WSDL 2.0

Web Service Description Language (WSDL), an XML-based specification, is the cornerstone of the Web service architecture, providing a common language for describing the services, and a platform for integrating those services. These service descriptions enable communication between the loosely-coupled services in an SOA-based cloud. A WSDL specification describes four critical pieces of information [61]:

i) interface information describing all the publicly available functions;

ii) data type information for all the message requests and responses;

iii) binding information about the messaging and transport protocol to be used; and

iv) address information for locating the specified service.

Web Service Description Language (WSDL) is currently available in two versions – WSDL 1.1 and WSDL 2.0. WSDL 1.1 was suggested as a language for describing Web Services in a note to W3C in March 2001 (by Ariba, IBM and Microsoft). The note also
described how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME types. The first working drafts of the next version, WSDL1.2, were published in 17 December, 2001 and its latest drafts were published on 11 June, 2003 [69]. WSDL1.2 was later renamed WSDL 2.0 due to major changes in WSDL1.1 and became a W3C Recommendation on 26 June 2007 [86][87][88].

As we have used WSDL 2.0 version in this thesis for describing services, the term ‘WSDL’ from this point onwards shall refer to this version only. For WSDL 1.1, the version would be specifically mentioned, wherever referred to.

Web Services Description Language Version 2.0 (WSDL 2.0) provides a model and an XML format for describing Web services. It describes a Web service at two fundamental stages – abstract and concrete – facilitating separation of the abstract details of ‘what’ functionality is offered by a service, from concrete details of a service description such as ‘how’ and ‘where’ that functionality is offered. Each stage, in turn, includes a number of constructs to promote reusability of the description and to separate independent design concerns. This separation of definitions helps to preserve the integrity of the service description regardless of the changes in the underlying technology platform [70][71].

At the abstract level, WSDL 2.0 describes a Web service in terms of the messages, operations, and interface. The messages that a service sends and receives are described independent of a specific wire format using a type system, typically XML Schema.

An operation associates a message exchange pattern with one or more messages. A message exchange pattern identifies the sequence and cardinality of messages sent and/or received as well as whom they are logically sent to and/or received from. An interface groups together operations without any commitment to transport or wire format.
At a concrete level, a binding specifies transport and wire format details for one or more interfaces. An endpoint associates a network address with a binding. And finally, a service groups together endpoints that implement a common interface.

A WSDL 2.0 service description indicates how potential clients are intended to interact with the described service. It represents an assertion that the described service fully implements and conforms to what the WSDL 2.0 document describes.

5.4.1. Elements of WSDL2.0

The top-level constructs in a WSDL document are types, interface, binding, service, import and include; and are contained in the description construct. A comprehensive description of the various major constructs in WSDL follows:

1. description – It is the root element in all the WSDL documents and serves as a container for all other service elements. It also includes the namespace declarations that would be used throughout the document. A required attribute of this element is targetNameSpace. It is a convention of XML Schema that enables the WSDL document to refer to itself. The value of this attribute is a URL. However, the namespace specification does not require the document to actually exist at this location; the important point is that the value specified is unique, different from all other namespaces that are defined.

2. types – This element encloses data type definitions used to define messages. It is used to impose constraints on the content of messages and faults. These constraints are based upon a specific data model, and expressed using a particular schema language. XML Schema may be used as schema language. There is only one types element associated with a service.
3. **interface** – This element describes sequences of messages that a service sends and/or receives. It does this by grouping related messages into operations. An operation is a sequence of input and output messages, and an interface is a set of operations. An interface can optionally extend one or more other interfaces. The set of operations available in an interface includes all the operations defined by the interfaces it extends directly or indirectly, together with any operations it directly defines. The operations directly defined on an interface are referred to as the declared operations of the interface. The name of the interface identifies it uniquely in a given namespace.

a. **operation** – This element is a child element of **interface** element and describes an operation that a given interface supports. An operation is an interaction with the service, consisting of a set of (ordinary and fault) messages exchanged between the service and the other parties involved in the interaction. The sequencing and cardinality of the messages involved in a particular interaction is governed by the message exchange pattern used by the operation. The various message exchange patterns endorsed by WSDL 2.0 are discussed in section 5.4.2. A message exchange pattern defines placeholders for messages, the participants in the pattern (i.e., the sources and sinks of the messages), and the cardinality and sequencing of messages exchanged by the participants. The ‘name’ and ‘message exchange pattern’ are the required attributes of this element. Each operation name is unique in a given namespace.
i. input and output – These elements define the content, or the payload, of a message exchanged in an operation. By default the message-content is defined by XML-based type system.

ii. infault and outfault – These elements define the fault message that may be input or output in an operation depending on the message exchange pattern.

b. fault – This is an optional child element of interface element. A fault is an event that occurs during the execution of a message exchange and disrupts the normal flow of messages. It is typically raised when a party is unable to communicate an error condition inside the normal message flow, or a party wishes to terminate a message exchange. It may also be used to communicate out of band information such as the reason for the error, the origin of the fault, as well as other informal diagnostics such as a program stack trace. Each fault element is uniquely identified by the name attribute.

4. binding – This element describes a concrete message format and transmission protocol which may be used to define an endpoint. It defines the implementation details necessary to access a service. A binding may be specified on a per-operation basis within an interface, or across all operations of an interface. The ‘name’ and ‘type’ are the required attributes of this element.

a. fault – This is a child element of binding element and describes a concrete binding of a particular fault within an interface, to a particular concrete message format, and is identified by a unique name.
b. operation - This is a child element of binding and describes the concrete message format(s) and protocol interaction(s) associated with a particular interface operation for a given endpoint. It is uniquely identified by the 'name' property.

i. input and output - These elements specify the concrete binding of input and output message participating in an operation to a particular concrete message format.

ii. infault and outfault - These elements define the concrete binding of fault message participating in an operation to a particular concrete message format.

5. service - This is a named construct that describes the endpoints at which a particular deployed implementation of the service is available. The required attributes of this element are 'name' and 'interface'.

a. endpoint - This is a child element of service element and defines the particulars of the endpoint (point of contact) at which a particular service is available. These elements are local to a given service. The required attributes of this element are 'name' and 'binding'.

6. include - This element allows separating the different components of a service definition, belonging to the same target namespace, into independent WSDL 2.0 documents. Specifically, it can be used to include components from WSDL 2.0 descriptions that share a target namespace with including description.

7. import - This element is used to import WSDL 2.0 components from a foreign namespace. This element has a required 'namespace' attribute.
5.4.2. Message Exchange Patterns

The operations in a service description are comprised, in part, of message definitions. The exchange of these messages constitutes the execution of a task represented by an operation. WSDL operations support different configurations of incoming, outgoing and fault messages; these configurations are referred to as patterns.

WSDL 2.0 supports eight different message exchange patterns (MEP) as follows [70][71][89][90]:

1. **in-out pattern** – This pattern consists of exactly two messages – the input message to the service is followed by a standard or a fault message output by the service. Here, the service requestor initiates the exchange of messages.

2. **out-in pattern** – This pattern also comprises of exactly two messages, but the output message from the service is followed by the input message to it. Here, the service provider initiates the exchange of messages. The input message may be a standard response message or a fault message from the requestor.

3. **in-only pattern** – This message exchange pattern consists of exactly one message. The service expects a single message and is not obligated to respond. The fault messages must not be propagated.

4. **out-only pattern** – This message exchange pattern consists of exactly one message. The service sends a single message. This is used primarily in support of event notification. The fault messages must not be propagated.

5. **robust in-only** – This message exchange pattern comprises of exactly one message, but provides the option of launching a fault message in the opposite direction as a result of a transmission or a processing error.
6. robust out-only – In this message exchange pattern, an outbound message initiates the transmission. Strictly, one message is exchanged and a fault message may be issued in response to the receipt of this message.

7. in-optional-out – This message exchange pattern consists of one or two messages. The input message to the service may be followed by an optional output message from it. This introduces the rule that the delivery of a response message is optional and should therefore not be expected by the service requestor. This pattern supports the generation of fault messages.

8. out-optional-in – This message exchange pattern comprises of one or two messages. The output message from the service may optionally be followed by an input message to the service.

5.4.3. Structural Differences between WSDL 1.1 and WSDL 2.0 documents

The key structural differences between WSDL1.1 and WSDL2.0 documents are listed in Table 5.1.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>WSDL 1.1</th>
<th>WSDL 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>The major elements of WSDL 1.1 document are definitions, types, message, portType, binding, service, documentation and import.</td>
<td>The major elements of WSDL 2.0 document are description, types, interface, binding, service, documentation, import and include.</td>
</tr>
<tr>
<td>2.</td>
<td>The root element is definitions.</td>
<td>The root element is description.</td>
</tr>
<tr>
<td>3.</td>
<td>Element portType present.</td>
<td>Element portType renamed interface.</td>
</tr>
<tr>
<td>S. No.</td>
<td>WSDL 1.1</td>
<td>WSDL 2.0</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>4.</td>
<td>Element port present.</td>
<td>Element port renamed endpoint.</td>
</tr>
<tr>
<td>6.</td>
<td>The import element enables importing WSDL definitions defined in separate files with the same or different namespaces.</td>
<td>The import element is used for WSDL definitions with a different namespace. While, include element is used for WSDL definitions with same namespace.</td>
</tr>
</tbody>
</table>

### 5.5. WSDL Meta Model

A metamodel in simple words is a ‘model of a model’. It describes the abstract syntax of the set of elements of a model. A WSDL metamodel, thus, is an explicit description of the elements (constructs and rules) of a WSDL model.

The description is the root element in a WSDL2.0 model and serves as a container for zero or more child elements in the following order [57]:

1. zero or more documentation element

2. zero or more elements from among the following, in any order:
   i. zero or more include element
   ii. zero or more import element

3. an optional types element
4. zero or more elements from the following, in any order:

A. interface element with zero or more elements from the following in any order:
   i. operation element with zero or more elements from the following in any order.
      a. input element
      b. output element
      c. input fault element
      d. output fault element
   ii. fault element

B. binding element
   i. operation element with zero or more elements from the following in any order.
      a. input element
      b. output element
      c. input fault element
      d. output fault element
   ii. fault element

C. service element with zero or more of the following element
   i. endpoint element
Figure 5.3 WSDL Metamodel
Figure 5.3 depicts the WSDL metamodel specified using a UML class diagram. The types and interface elements which constitute the abstract description are technology-independent, while the binding and service elements which constitute the concrete description are technology-specific. The message exchange pattern for an operation may be any one of the eight enumerated in the metamodel in Figure 5.3.

5.6. Mapping Service PIM to WSDL PSM

Once the PIM of the cloud software service is specified, it can be transformed into a WSDL PSM based on the transformation definition specified for the purpose. An automated or a semi-automated transformation tool may be used to execute the transformation definition.

The transformation definition that specifies the transformation rules for mapping the constructs of the service PIM into the constructs of WSDL PSM is simple and straightforward. The key mappings of the transformation definition are listed below:

i. The interface of the service in the PIM maps to the interface element of WSDL PSM.

ii. The operations in the PIM map to the operation element in the WSDL PSM.

iii. The data types in the PIM map to the types element of WSDL PSM.

iv. Each input message in the PIM maps to the input element in the operation element of the WSDL PSM.

v. Each output message in the PIM maps to the output element in the operation element of the WSDL PSM.
5.7. Modeling Cloud SaaS Interoperability

This section pictorially represents the interoperability between the two cloud services, OHRS_Service and COHS_Service, illustrated in section 4.6. The UML class diagram has been used to model the PIMs of these two services as depicted in Figure 5.4.

An interface, in a service, is a collection of operations that have no corresponding method implementations. It is simply a contract specifying the signatures of the methods it includes. In UML, the interface may be represented using a ball-and-socket notation or a stereotype notation. An interface cannot instantiate itself. It must be realized by a class in order for the implementations of its operations to be made visible. Figure 5.4 depicts the COHS_Service interface using the "ball" notation. This interface is realized by associating it with the COHS_Service class. As mentioned earlier, the services in an SOA-based cloud are loosely-coupled so that they may evolve independently without forcing any change in other parts of the system. The interfaces thus become significant here. The services interact with each other through provided and required interfaces. The provided interface of a service, which is represented as a 'ball', describes the functionalities offered by the service. On the other hand, the required interface of a service, which is represented as 'socket', declares that it requires the functionalities
offered by another service [17]. In Figure 5.4, the COHS_Service has a provided interface associated with it, while the OHRS_Service has a required interface associated with it. Also, as the OHRS_Service is a client in this example, the OHRS_Service class has been stereotyped, based on UML profile, as a ‘Service Client’.

5.8. Formalizing PIM to WSDL PSM Transformation

Definition

The transformation definition for transforming a cloud service PIM to cloud service PSM targeted on Web Service platform as discussed in Section 5.6 has been formalized by the authors using a transformation tool. The transformation tool is implemented in Java and has been developed using NetBeans IDE. This transformation tool comprises of web pages at the front end, enabling the user to retrieve or display information, and the transformation logic running at the back end. The web pages have been created using Java Server Pages (JSP) and the logic behind the application has been implemented using Java HTTP Servlets [91][92][93][94][95].

NetBeans IDE is a sophisticated open-source integrated development environment. It provides all the tools necessary to develop desktop, enterprise, web and mobile applications. The applications can be executed on any operating system where a compatible JVM is running; this includes Microsoft Windows, Mac OS, Linux and Solaris. Besides, these applications in NetBeans are a set of modular software components and can be extended by third party developers.

A Java servlet is a platform-independent, server-side Java program. It is basically a Java .class file that helps to extend the functionality of a Web server. Even though the servlets are written in Java, their clients may not necessarily be written in Java. They are used in
the middle tiers of distributed application systems. Servlets can in turn be clients to other services, written in some other languages. Although, the servlets are not tied to a specific client-server protocol they are most commonly used with HTTP and the word servlet is often used to represent HTTP Servlet. The servlets run within a Web server's namespace and therefore do not display a graphical interface to the user. They run behind the scene on a Web server and provide a framework for creating applications that implement a request/response paradigm.

Java Server Pages (JSP) provides web and Java developers a simple yet powerful framework for creating web applications. The JSP framework provides the language for developing web pages and for accessing server-side objects. The web content can be created with both static and dynamic components. Also, it provides the developers with the ability to encapsulate and separate program logic from the presentation logic.

Thus, while JSP focuses on simplifying the creation and maintenance of the HTML, servlets are best at invoking the business logic and performing complicated operations.

In this transformation tool, the web pages with text boxes, combo boxes and buttons provide a graphical interface to the user to input the various parameters of the PIM of cloud SaaS. The logic, which represents the transformation definition, is implemented using Java HTTP servlets. The output of the execution of transformation logic is a .wsdl document that represents the WSDL PSM.

Annexure B displays the snapshots of PIM to WSDL PSM transformation process executed using the transformation tool developed by the authors.

The pseudocode of transformation logic for transforming the cloud service PIM into corresponding WSDL PSM may be expressed as:

1. Start.
2. Input service name, WSDL document target name space.
3. Input service URI.
4. Input interface name.
5. Input schema definition target name space (URI).
6. Input operation name.
7. Select message exchange pattern (mep).
10. If mep = “in-only”, execute in-only method. Go to step 16.
11. If mep = “out-only”, execute out-only method. Go to step 16.
16. To add another operation go to step 6 else step 17.
17. Input binding name.
18. Select message protocol.
19. Select transmission protocol.
21. Stop.

**IN-OUT MEP METHOD**

1. Set mep URI = "http://www.w3.org/ns/wsdl/in-out".
2. Input input_parameters and their data types.
3. Input output_parameters and their data types.
4. Input fault name and its data type.
5. Return.

**OUT-IN MEP METHOD**

1. Set mep URI = "http://www.w3.org/ns/wsdl/out-in".
2. Input output_parameters and their data types.
3. Input input_parameters and their data types.
4. Input fault name and its data type.
5. Return.

**IN-ONLY MEP METHOD**

1. Set mep URI = "http://www.w3.org/ns/wsdl/in-only".
2. Input input_parameters and their data types.
3. Return.

**OUT-ONLY MEP METHOD**

1. Set mep URI = "http://www.w3.org/ns/wsdl/out-only".
2. Input output_parameters and their data types.
3. Return.
A Model-Driven Approach to Cloud SaaS Interoperability

ROBUST-IN-ONLY MEP METHOD

1. Set mep URI = "http://www.w3.org/ns/wsdl/robust-in-only".
2. Input input_parameters and their data types.
3. Input fault name and its data type.
4. Return.

ROBUST-OUT-ONLY MEP METHOD

1. Set mep URI = "http://www.w3.org/ns/wsdl/robust-out-only".
2. Input output_parameters and their data types.
3. Input fault name and its data type.
4. Return.

IN-OPTIONAL-OUT MEP METHOD

1. Set mep URI = "http://www.w3.org/ns/wsdl/in-optional-out".
2. Input input_parameters and their data types.
3. Input output_parameters and their data types (optional).
4. Input fault name and its data type (optional).
5. Return.

OUT-OPTIONAL-IN MEP METHOD

1. Set mep URI = "http://www.w3.org/ns/wsdl/out-optional-in".
A Model-Driven Approach to Cloud SaaS Interoperability

Figure 5.5 Work Flow Diagram of PIM to WSDL PSM Transformation
2. Input output parameters and their data types.
3. Input input parameters and their data types (optional).
4. Input fault name and its data type (optional).
5. Return.

Figure 5.5 depicts the workflow diagram for generating the WSDL PSM from PIM.

5.9. Conclusion

Web service is a fundamental technology underlying the Cloud computing paradigm; and is evolving too. The transitions in web technology have more serious implications as business-to-business interactions are involved in it. In order to tackle this issue, based on MDA approach, a formal, semantically rich platform independent model of the cloud software service (Web service) capturing the information and functionalities provided by it, may be defined which may then be used to generate the artifacts that support the service over some other set of technologies.

WSDL is the core component of the Web service technology which is responsible for defining the service interface. In this chapter, the PIM of the cloud service is transformed into a WSDL PSM using a transformation tool developed by us. This WSDL PSM specifies the platform specific details of the cloud service interface. This PSM, approximates the code, and can be invoked by other cloud software services. SOAP, a messaging and communication protocol, is being used for the exchange of messages between the services. WSDL itself has the ability to bind to SOAP, simple HTTP or MIME. WSDL, which represents the PSM level of the MDA-based cloud SaaS, thus ascertains interoperability among the cloud SaaS.