Towards a MaaS Service for Cloud Service Interoperability

Nour El Houda Bouzerzour[1,2][a] and Yahya Slimani[1,3]

[1] *LISI Laboratory of Computer Science for Industrial Systems, Carthage University, Tunis, Tunisia*

[2] *ENSI, La Manouba University, Manouba, Tunisia*

[3] *ISAMM, La Manouba University, Manouba, Tunisia*

Keywords: Cloud Computing, Cloud Service Description, Service Interoperability, Transformation Rules, ATL, WSDL.

Abstract: Cloud computing is an emerging computing paradigm, which provides high service availabality, high scalability as well as low usage costs. This has encouraged enterprises and individual users to embrace cloud technology. However, the lack of service interoperability (also known as the vendor lock-in) issue still persists. The vendor lock-in is caused by the cloud service providers who aim to prevent the clients from switching to other clouds or providers. The solutions to overcome the vendor lock-in addressed a specific cloud actor or a specific cloud model, which makes them not generic. Thus, we present in this paper, our Cloud Interoperability Pivot Model (CIPiMo). CIPiMo is a Model-as-a-Service, which standardizes the cloud service description languages by transforming them into a Generic Cloud Service Description model (GCSD) to make them interoperable. We rely on MDE techniques to achieve a Model-to-Model transformation. Therefore, we define mappings between the source description languages (OWL-S and WSDL) and the target language (GCSD). Furthermore, we illustrate our proposed meta-models for each language and we implement our transformations using ATL with OCL. Eventually, we use a static analyzer (AnATLyzer IDE) to validate the correctness of our transformations. We provide use cases to demonstrate the applicability of our approach.

1 INTRODUCTION

Cloud computing has known a huge spread and popularity among providers and clients. The reason being that cloud computing offers an on-demand access to a pool of shared computing resources (Liu et al., 2011) with high availability and scalability, as well as in a "pay-as-you-go" fashion. Despite the numerous advantages of cloud computing, providers and clients are still hesitant to adopt it because of its prominent obstacles. The security and privacy concerns, the service recovery from disastrous situations and the SLA management are among these obstacles. However, the most prominent issue is the lack of cloud service interoperability. The latter is defined as the ability of heterogeneous systems to communicate, whether they are deployed on the same cloud or on multiple clouds (Opara-Martins et al., 2014). The cloud providers, who offer proprietary services and decline standardization, are causing the service interoperability issue. This is also known as the vendor lock-in, where the providers prevent the clients from switching from one provider to another or from one cloud to another in interconnected cloud environments (Opara-Martins et al., 2014). The solutions that were proposed in the literature (such as brokers, standards and semantic approaches) either target a single cloud actor (client or provider), they are specific to a certain technology, or they do not address all the cloud models (SaaS, PaaS, IaaS). Therefore, these approaches are not generic (Bouzerzour et al., 2020b). Other solutions such as RESTful APIs and gRPC are widely adopted to enable systems' interoperability. However, they are still faced with issues. The semantic REST APIs are not adopted or recognized by commonly used tools (Cheron et al., 2019). Furthermore, the communication between two cloud systems using different protocols (e.g: REST HTTP protocol and MQTT *(Message Queuing Telemetry Transport)* protocol) requires a protocol adapter to happen. Eventually, The use of RESTful APIs does not solve the semantic inconsistencies, which are created by adopting heterogeneous cloud systems' descriptions (Baudoin et al., 2014).

Therefore, in this paper, we propose a cloud service interoperability approach based on the standardiza-

[a] https://orcid.org/0000-0002-8785-6631

Bouzerzour, N. and Slimani, Y.
Towards a MaaS Service for Cloud Service Interoperability.
DOI: 10.5220/0010911400003119
In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 72-83
ISBN: 978-989-758-550-0; ISSN: 2184-4348

tion of cloud service descriptions. Thus, we define the following research hypothesis:
RH1: the heterogeneity of cloud service descriptions is among the reasons causing the vendor-lock-in,
RH2: standardizing service descriptions will allow their interoperability.
To enable the standardization of cloud service descriptions, the authors of (Ghazouani and Slimani, 2017) proposed a Generic Cloud Service Description model (GCSD). The GCSD describes cloud services in a standardized and comprehensive manner (from multiple aspects), which was demonstrated by use case scenarios that describe different services from different delivery models (SaaS, PaaS, IaaS) (Ghazouani and Slimani, 2017). Therefore, we rely on an MDE model-to-model transformation technique to transform heterogeneous description languages to the GCSD to make them standardized and interoperable. MDE is a software engineering approach, which uses human and machine-readable models. MDE considers models as first class entities to express specifications of systems at different levels of abstraction. The model's elements, the relations between them and their constraints are described by a meta-model (Bezivin et al., 2005). We rely on MDE because it is expected to gain more growth in the software industry, as it was stated and approved by various high quality research and studies that MDE is indeed able to provide effective and helpful solutions to improve the software development process (Brambilla et al., 2017).

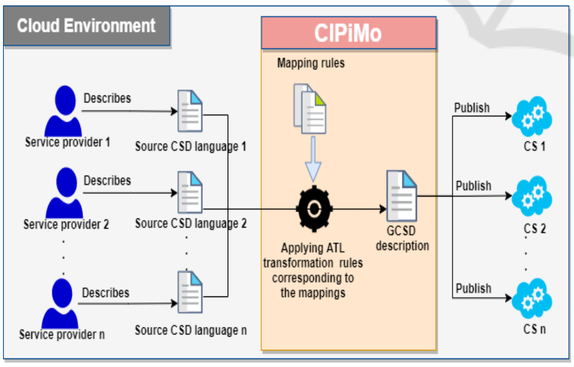

Figure 1: Our proposed interoperability model CIPiMo (Bouzerzour et al., 2020a).

In this paper, we present an overview of our interoperability model, which we name CIPiMo *(Cloud service **I**nteroperability **Pi**vot **Mo**del)*. Furthermore, we implement, validate, and we test our MaaS pivot model by providing use cases for different description languages.
The rest of this paper is organized as follows: In Section 2, we present our MaaS pivot model CIPiMo. Whereas is Section 3, we present the mappings of the source description languages to the GCSD as well as their meta-models. In Section 4 we implement our transformation rules, and we discuss the results. Section 5 discusses the static validation of our transformations using AnATLyzer. In Section 6, we present the related works and in Section 7 we conclude this paper.

2 CLOUD SERVICE INTEROPERABILITY PIVOT MODEL (CIPiMo)

CIPiMo is an MDE-based MaaS (Model as a Service) service (aka. Modeling as a Service), which enables cloud service interoperability by transforming heterogeneous service descriptions into a GCSD. MaaS is a SaaS variant *(Software as a Service)*, which allows users to deploy and execute model-driven and modeling services over the Internet, and it provides an interface for the client to communicate with the services. Among the main contributions of MaaS is enabling interoperability between tools and systems by bridging the gaps between their specifications (Bruneliere et al., 2010). Thus, CIPiMo aims to overcome the vendor lock-in by promoting: (i) client-centric interoperability, which allows enterprises or end-users to adopt a multi-cloud strategy to interoperate their services, which are deployed on different providers or clouds, and it encourages enterprises to migrate their legacy systems to the cloud (Bouzerzour et al., 2020b); and (ii) provider-centric interoperability, which enables cloud federations between SME *(Small and Medium Enterprises)* service providers to gain more computing power and scalability, or hybrid cloud strategy for bursting the resources at peak moments (Bouzerzour et al., 2020b). This transformation mediator maps the concepts and elements of source languages to the GCSD concepts. Then, it applies a set of transformation rules to transform the source language into the generic description. Hence, the service descriptions will be unified and interoperable. We chose a transformation through a pivot rather than a direct transformation from a source language to a target language because it requires fewer transformations, especially if the number of languages to be transformed is substantial (Boukhari et al., 2012). Figure 1 presents an overview of CIPiMo: in the cloud environment, service providers describe their services using different CSD languages (such as OWL-S and WSDL), which will be transformed using CIPiMo by applying the transformations corresponding to the proposed mappings for each language. Then, the resulting descriptions will be standardized and ready to be published. The mapping phase is achieved by the manual extraction of different elements and their corresponding definitions from the corresponding specification document of each language. Then, we align the source element with its equivalent target element.

To demonstrate the functionality of our proposed pivot model we provide use cases, which transform WSDL and OWL-S service descriptions to the GCSD. We fully recognize that WSDL and OWL-S were not originally created for describing cloud services. However, both of these languages were used by researchers, who considered cloud services as web services, to describe cloud services (Ghazouani and Slimani, 2017). In (Zhou et al., 2011), the authors used WSDL-S for semantically describing services for SaaS discovery. Whereas, in (Goscinski and Brock, 2010), a WSDL file extension was proposed to take into consideration cloud characteristics. OWL-S was used in (Martino et al., 2014) to semantically describe Microsoft Azure API functional and non functional properties and it was also used in (Karim et al., 2014) to define cloud services and their Quality of Service (QoS). Moreover, the available cloud service description and modeling approaches that were presented in the literature (such as (Bergmayr et al., 2014) (Perez and Rumpe, 2014) (Andrikopoulos et al., 2014)) addressed the description of the deployment, the configuration, and the provisioning of the cloud service rather than the description of the service properties and functionality. Furthermore, the majority of the cloud description and modeling languages lack formal documentation or specification documents to define in details their constructs and elements. Unlike that, WSDL and OWL-S are thoroughly documented in their specification documents (Christensen et al., 2001) (Martin et al., 2004). Moreover, many legacy systems were based upon these languages and they provide all the basic required concepts to describe a service and to demonstrate our proposal. Therefore, in the aim of demonstrating the genericity of our pivot model, we chose to run our interoperability model on WSDL (which is a syntactic language) and on OWL-S (which a semantic language), respectively, to unify their description and make them interoperable.

3 PROPOSED MAPPING RULES

In this section we present the mappings from WSDL and OWL-S to the GCSD. Furthermore, we illustrate our proposed exhaustive meta-models for each language. To model the description languages meta-models we use Eclipse Modeling Framework (EMF) ECORE meta-model as the de-facto standard modeling framework in the industry.

3.1 Mapping WSDL to GCSD

WSDL *(Web Services Description Language)* (Christensen et al., 2001) is an XML format to describe web services as collections of network endpoints operating on messages. To define network services, WSDL 1.1 uses, mainly, six elements, which are Types, Message, PortType, Binding, Port; and Service. However, WSDL only describes services from a technical aspect. Figure 2 depicts, our proposed WSDL meta-model, which illustrates all WSDL elements required to describe a service and the relations between them. Table 1 depicts the proposed mapping from WSDL to GCSD.
As depicted in Table 1, the WSDL `operation` element, which describes the operations that are performed by the service, is transformed into the `Function` concept in the GCSD, which in its turn describes a course of actions to be performed by the service. Each message `part` element is transformed into a `Parameter` instance as they both describe the parameters required by the `operation/Function`. The `input` and `output` are also mapped to `Parameter` concept because they define the abstract messages formats. `PortType` describes a set of abstract operations and it describes the abstract message that is involved in the operation. Therefore, it is mapped to the abstract class `Interface`. `Binding`, which defines a concrete protocol for the operations defined in a portType is mapped to `Protocol`. The `operation name` and the `part name` are mapped to the concept `Description`, which attaches typed textual descriptions to the other concepts (in this case, names are Description objects of type name). `Import location, port location`, and `definition targetNameSpace` are all mapped to `Artifact URI`, which represents a link to a resource that can be located using a URI.

We do not define mappings for `message` because it provides an abstract definition of the data being transmitted by its parts; it is presented as an argument, which is mapped to a method invocation. Therefore, the mapping of the different `part` elements of the enclosing message implies the mapping of the `message` element.

Table 1: Mapping WSDL elements and OWL-S classes into the GCSD concepts.

WSDL elements	**Description**	**GCSD concepts**	
definition	It is the root element of all WSDL documents. It defines the name of the web service.	-	GCSDservice (root concept)
service	A service groups a set of related ports together	ServiceModule	Service
part	It defines the parameters of the web service function.	FunctionalModule	Parameter
operation	It is a transmission primitive that an endpoint can support.		Function
input	It specifies the abstract message format for an operation.		Parameter
output	It specifies the abstract message format for an operation.		Parameter
fault	It specifies the abstract message format for any error messages that may be output as the result of the operation.		Fault
portType	It is a set of abstract operations.	TechnicalModule	Interface
binding	It specifies the concrete protocol and data format specifications for the operations and messages defined by a particular portType.		Protocol
soap:binding: transport	It indicates the SOAP transport that the binding corresponds to.		AccessProfile
operation:name	It is an operation name, which is not required to be unique.	FoundationModule	Description: value
types	It is a data type definitions used to describe the messages exchanged.		TypeReference
import: location	It associates a namespace with a document location.		Artifact:URI
port: location	It is an address for a binding and it defines a single communication endpoint.		Artifact: URI

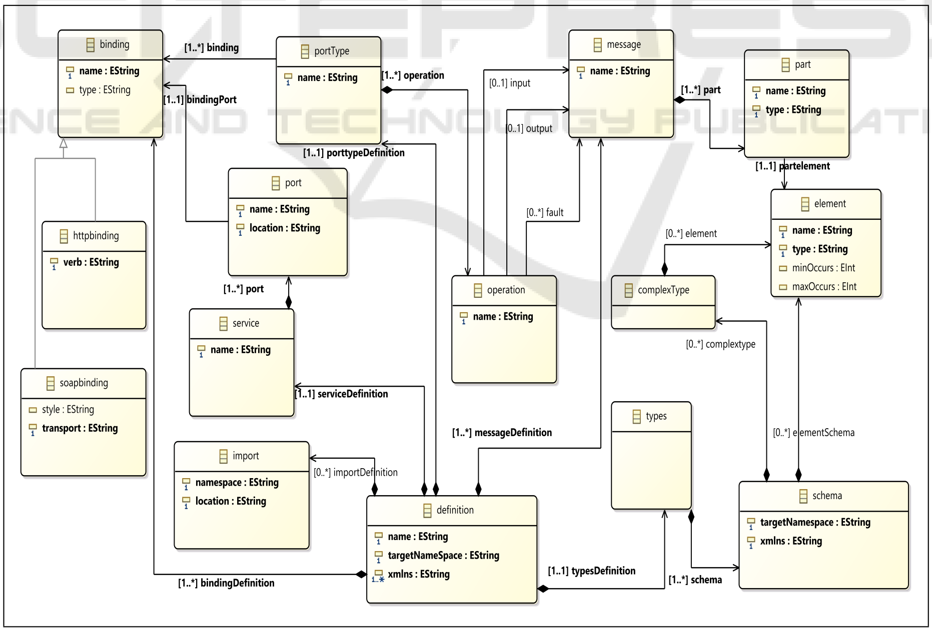

Figure 2: Our proposed WSDL meta-model.

3.2 Mapping OWL-S to GCSD

OWL-S (Martin et al., 2004) is an ontology for web service description, which is based on three sub-ontologies: (i) serviceProfile, (ii) serviceModel; and, (iii) Grounding. However, it lacks the business aspect of the service. Our proposed OWL-S meta-model is depicted in Figure 3. Whereas, the mappings of OWL-S classes to GCSD concepts were detailed in (Bouzerzour et al., 2020a).

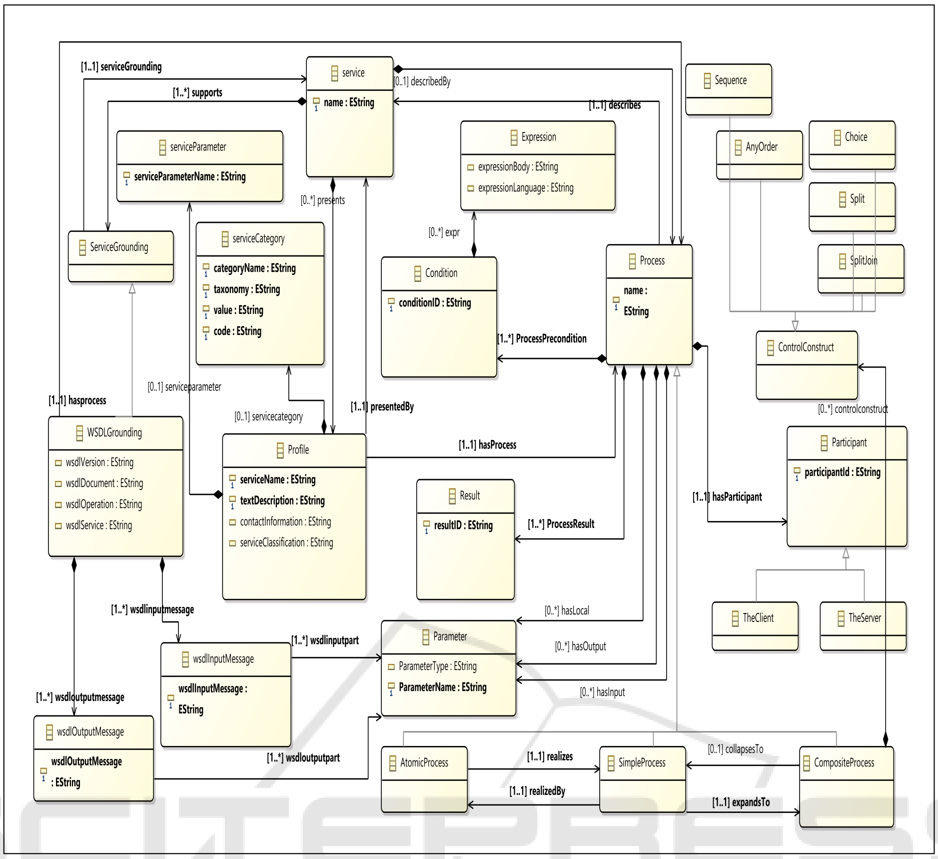

Figure 3: Our proposed OWL-S meta-model.

We also define a meta-model for the GCSD. Given that the GCSD is based on Unified Service Description Language (USDL) (Barros and Oberle, 2012), it has nine concepts to represent USDL's modules. Furthermore, each concept regroups sub-concepts. Therefore, we do not include it the GCSD meta-model in this paper for space reasons.

4 IMPLEMENTING TRANSFORMATION RULES

We rely on Atlas Transformation Language (ATL) (Jouault et al., 2008) to implement the transformation rules. ATL is a domain-specific language for unidirectional Model-to-Model (M2M) transformations, which provides declarative and imperative constructs. Moreover, ATL is built upon Object Constraint Language (OCL) (Cabot and Gogolla, 2012), which is an OMG standard and a typed declarative language.

Therefore, in this section we present our implementation using ATL and we define OCL invariants for WSDL and OWL-S. However, we do not define any constraints for the GCSD, as it is based on USDL, which does not specify any constraint to allow openness and genericity. Otherwise, specifying constraints for specific types will result in enlarging USDL core model with the complex and overlapping constraints (Barros and Oberle, 2012). To implement our transformations we use ATL toolkit (Version 4.2.0) on top of Eclipse IDE (Version: 2020-12 (4.16.0)). We also use ATLauncher (Guana, 2015) to programmatically launch our transformations. ATLauncher is standalone Java class that runs ATL M2M transformations outside the Eclipse to promote the integration of MDE tools with other software engineering solutions. Therefore, using ATLauncher, our transformations will be integrated into a JAVA application, which will be offered as a MaaS service to allow cloud service interoperability.

4.1 Transformation Rules from WSDL to GCSD

Figure 4 depicts the implemented transformation rules from WSDL to GCSD and Figure 5 presents the OCL constraints that we implemented for our transformation.

```
module wsdl2gcsd;
create OUT: gcsd from IN: wsdl;
rule definition2GCSDservice {
     from s: wsdl!definition(wsdl!definition.allInstances()->forAll(s1, s2 |s1 <> s2
                          implies s1.name <> s2.name)
     and wsdl!definition.allInstances()->forAll(m | m.name.size() <= 1 ))
     to t : gcsd!GCSDservice (name <- s.name),
     t1 : gcsd!Artifact (URI <- s.targetNameSpace),
     t2 : gcsd!Description (scope <- 'targetNameSpace', type <- #freetextShort) }
rule import2Artifact {
    from s: wsdl!import
    to  t:gcsd!Artifact (URI <- s.location) }
rule part2Parameter{
    from s: wsdl!part,
        s1: wsdl!schema
    to  t:gcsd!Parameter (sampleValues <- 'Param'),
        t1:gcsd!Description  (value<- s.name,   scope <- 'parameterName', type<- #name),
        t2:gcsd!TypeReference (unitSymbol<- s.type, classificationSystemID <- s1.xmlns ),
        t3: gcsd!Artifact ( URI <- s1.targetNamespace),
        t4:gcsd!Description (scope <- 'targetNameSpace', type<- #freetextLong) }
rule Operation2Function {
    from  s1: wsdl!operation
    to  t: gcsd!Function (inputs <- s1.input, outputs <- s1.output, faults <- s1.fault),
        t1:gcsd!Description  (value<- s1.name, scope <- 'functionName', type<- #name) }
rule service2Service {
    from s : wsdl!service
    to t: gcsd!Service (serviceName <- s.name) }
rule soapbinding2Protocol {
    from s: wsdl!soapbinding
    to t: gcsd!Protocol  (identifier <- s.transport) }
rule PortType2Interface{
    from s: wsdl!portType,
        s1: wsdl!port
    to  t: gcsd!Interface(implementationSpecification <-s1.bindingPort, implementationTypeId <-s.name),
        t1: gcsd!Artifact (URI <- s1.location) }
```

Figure 4: WSDL transformation rules.

```
The service name must be unique among all services defined within the enclosing WSDL document.
        -- @pre wsdl!service.allInstances()->forAll(s1,s2| s1<>s2 implies s1.name<>s2.name)
The port name must be unique among all ports defined within the enclosing WSDL document.
        -- @pre wsdl!port.allInstances()->forAll(s1| s1<>self implies s1.name<>self.name)
A port must not specify more than one address.
        -- @pre wsdl!port.allInstances()->forAll(s1| s1<>self implies s1.location<>self.location)
The binding name must be unique among all bindings defined within the enclosing WSDL document.
        -- @pre wsdl!binding.allInstances()->forAll(s1,s2| s1<>s2 implies s1.name<>s2.name)
The portType name must be unique among all portTypes defined within the enclosing WSDL document.
        -- @pre wsdl!portType.operation->forAll(s1,s2| s1<>s2 implies s1.input<>s2.input)
The part name must be unique among all parts defined within the enclosing WSDL document.  If the part
name attribute is not specified on a message, it defaults to the name of the operation.
        -- @pre if part.name.oclIsUndefined()  then part.name= operation.name  else
        part.allInstances()->forAll(s1,s2| s1<>s2 implies s1.name<>s2.name) endif
```

Figure 5: OCL constraints for WSDL transformation.

As it is shown in Figure 4, rule definition2GCSDservice transforms the WSDL `definition`, which is the root element of the service description to `GCSDservice` concept with the definition `name` represented by the GCSDservice `name` and the `targetNamespaces` is represented by the `Artifact` concept' attribute `URI`. The constraint states that a definition may have an optional but unique name. Next, rule import2Artifact transforms the `import` element to the `Artifact` concept, which provides a `URI` property to locate a resource. The rule part2Parameter depicts the transformation of the WSDL `part` class and its `type` to `Parameter` concept and `TypeReference` concept, which represents the parameter type using `unitSymbol` property. The part `name` is transformed to the `Description` concept's `value` attribute. Then, the rule Operation2Function transforms WSDL `operation` to `Function` concept. The operation's `input, output and fault` properties are described by the GCSD relations `inputs, outputs, and faults`, respectively; and the operation `name` is described by the `value` property of the Description concept.

For the concrete description of WSDL services, the rule service2Service transforms of the `service` class and the service `name` to the GCSD `Service` concept and the property `serviceName` respectively. The rule soapbinding2Protocol transforms the binding `transport` to the `Protocol` concept `identifier`. Eventually, the PortType2Interface rule transforms WSDL `portType and port` elements to `Interface` concept, the `port binding` is transformed to Interface's `implementation-`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl" xmlns:tns="http://example.com/stockquote.wsdl"
xmlns:xsdl="http://example.com/stockquote.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types> <schema targetNamespace="http://example.com/stockquote.xsd" xmlns="http://www.w3.org/2000/10/XMLSchema">
            <element name="TradePriceRequest" type="string"/>
            <complexType> <all> <element name="price" type="float"/> </all> </complexType>
    </schema> </types>
    <message name="GetLastTradePriceInput">
        <part name="inputbody" element="xsdl:TradePriceRequest"/>
    </message>
    <message name="GetLastTradePriceOutput">
        <part name="outputbody" element="xsdl:price"/>
    </message>
    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
            <input message="tns:GetLastTradePriceInput"/>
            <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>
    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
            <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
            <input> <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
    </binding>
    <service name="StockQuoteService">
        <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
            <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>
</definitions>
```

Figure 6: WSDL description of StockQuoteService service.


```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:gcsd="www.gcsd.com"  xmlns:theWSDLecore="http://www.example.org/theWSDLecore">
  <gcsd:GCSDservice name="StockQuote"/>
  <gcsd:Artifact URI="http://example.com/stockquote.wsdl"/>
  <gcsd:Description scope="targetNameSpace" type="freetextShort"/>
  <gcsd:Artifact URI="http://example.com/stockquote/schema"/>
  <gcsd:Parameter sampleValues="Param"/>
  <gcsd:Description value="inputbody" scope="parameterName"/>
  <gcsd:TypeReference unitSymbol="string"/>
  <gcsd:Artifact URI="http://example.com/stockquote.xsd"/>
  <gcsd:Description scope="targetNameSpace" type="freetextLong"/>
  <gcsd:Parameter sampleValues="Param"/>
  <gcsd:Description value="outputbody" scope="parameterName"/>
  <gcsd:TypeReference unitSymbol="float"/>
  <gcsd:Artifact URI="http://example.com/stockquote.xsd"/>
  <gcsd:Description scope="targetNameSpace" type="freetextLong"/>
  <gcsd:Function>
    <inputs xsi:type="theWSDLecore:message" name="GetLastTradePriceInput">
      <part name="inputbody" type="string">
        <partelement href="../transformation/now.xmi#//@typesDefinition/@schema.0/@elementSchema.0"/>
      </part>
    </inputs>
    <outputs xsi:type="theWSDLecore:message" name="GetLastTradePriceOutput">
      <part name="outputbody" type="float">
        <partelement href="../transformation/now.xmi#//@typesDefinition/@schema.0/@complextype.0/@element.0"/>
      </part>
    </outputs>
  </gcsd:Function>
  <gcsd:Description value="GetLastTradePrice" scope="functionName"/>
  <gcsd:Service serviceName="StockQuoteService"/>
  <gcsd:Interface implementationTypeId="StockQuotePortType">
    <implementationSpecification xsi:type="gcsd:Protocol" identifier="http://schemas.xmlsoap.org/soap/http"/>
  </gcsd:Interface>
  <gcsd:Artifact URI="http://example.com/stockquote"/>
</xmi:XMI>
```

Figure 7: StockQuoteService service description in GCSD.

`Specification` and the portType `name` is transformed to the `implementationTypeId` property, which references the service interface.

To test our transformation, we apply our trans-

A *service* can be described by at most one service model.

```
--@pre owls!service.allInstances()->forAll(b|b.describedBy->size()=1)
```

A *grounding* must be associated with exactly one service.

```
--@pre owls!ServiceGrounding.allInstances()->forAll(b|b.serviceGrounding->size()=1)
```

Figure 8: OCL constraints for OWLS- transformation.

formation rules on WSDL *"StockQuote"* service instance, which is illustrated in figure 6. After applying our transformation, we get the following description in GCSD (XMI file) (as shown in Figure 7). As depicted in Figure 7, the `GCSDService` is the main concept, which is located in the `URI` (described by the `Artifact`). The second Artifact URI describes the WSDL schema nameSpace. Two `Parameters` are described, with their names (described by `Description`), their type (described by `TypeReference`), the type xsd schema URI (described by `Artifact`). One `Function` is described with its inputs and outputs and name (described by `Description`). A `Service` is described with its name and an `Interface` is also described with its implementationSpecification and its URI (described by `Artifact`).

4.2 Transformation Rules from OWL-S to GCSD

Figure 8 presents the OCL constraints that we implemented for our transformation, and Figure 9 depicts the implemented transformation rules from OWLS to GCSD.

The rule service2GCSDservice transforms OWL-S `service` and `Profile` classes into `Service` and `ContactProfile` concepts to describe the service's name and general information. The rule Process2Function transforms OWL-S `Process` (of type AtomicProcess) and `Participant` (of type TheClient) into `Interaction`, `Function` and `Consumer` concepts, respectively. The `hasParticipant` property is transformed into `involvedRoles` to describe the participant interacting with the process and it also specifies the `roleDescription` for the Consumer concept. Whereas, the `hasInputs, hasOutput, ProcessPrecondition, and ProcessResult` properties are transformaed into `input, outputs, preconditions, and postconditions`, respectively. The Paramter2Parameter rule transforms OWL-S `Parameter` and its type (described by `ParameterType` attribute) into GCSD `Parameter` and `TypeReference` (which describes the type using `unitSymbol` attribute) concepts.

The precondition2precondition rule transforms OWL-S `Condition` and `Expression` into `Condition` concept, the OWL-S condition's expression (described by `expr` property) is transformed into the `conditionExpression` property. The OWL-S condition's expression body (`expressionBody` attribute) and expression language (`expressionLanguage` attribute) are transformed into GCSD Expression `value` and `languageID`, respectively. The rule result2postcondition transforms the OWL-S `Result` into GCSD `Condition` and the `resultID` attribute into Description's `value` attribute. The rule grounding2Interface transforms OWL-S `grounding` into `Interface` concept and the grounding `wsdlDocument` attribute is transformed into `implementationTypeId` attribute. The rule theServer2provider transforms OWL-S participant of type `theServer` into GCSD `Provider`. Eventually, the rule CompositeProcess2Phase transforms OWL-S Process of type `CompositeProcess` into GCSD `Phase` and the Process `name` is transformed into Description's `value` attribute.

We apply our transformation rules on OWL-S *"CongoBuyService"* service instance, as shown in Figure 10. The resulting description in GCSD is depicted in Figure 11. The `GCSDservice` describes the OWL-S service and its name. The `capabilities` describe the Process achieved by the OWL-S service. `ContactProfile` provides the contact information (described by `Description`). The `Interaction` describes the participants involved in the atomicProcess and the `Process` is described by the `Function` and its inputs, outputs, preconditions, and postconditions. `Consumer` describes the participants and their types (client or provider). Two `Parameters` are described with their types and names (described by `TypeReference and Description`) and one `Condition` is described alongside its `Expression` body and language. The `Interface` describes the WSDL document for the grounding using the `implementationTypeId`. OWL-S CompositeProcess is composed of subprocesses (atomic and composite) and it is constructed using control constructs and references to processes called PERFORMs. Therefore, the transformation of OWL-S CompositeProcess requires defining UML activity for describing the control constructs, UML sequence diagram for describing the compositions, and OCL to specify the pre- / post-conditions required for the composition (Bouzerzour et al., 2020a). Thus, this transformation is out of the scope of this paper, and it will be included in a future work.

```
module owlsTrans;
create OUT : gcsd from IN : owls;
    rule service2GCSDservice {
        from s: owls!service, s1:owls!Profile
        to t:gcsd!GCSDservice ( name<-s.name),
            t4: gcsd!Service (capabilities<-s1.hasProcess),
            t2: gcsd!ContactProfile,
            t3:gcsd!Description (value<- s1.contactInformation, scope <- 'contact information', type<- #name) }
    rule Process2Function {
        from s: owls!AtomicProcess , s1: owls!Participant(s1.oclIsTypeOf(owls!TheClient) and s.oclIsTypeOf(owls!AtomicProcess) )
        to t: gcsd!Interaction (involvedRoles <- s.hasParticipant),
           t1: gcsd!Function (inputs<-s.hasInput, outputs<-s.hasOutput, preconditions <- s.ProcessPrecondition, postconditions <-s.ProcessResult),
           t2:gcsd!Description (value<- s.name, scope <-'interactionName', type <- #name),
           t3: gcsd!Consumer (roleDescription <-s.hasParticipant),
           t4:gcsd!Description (value<- s1.participantId, scope <- 'TheConsumerId', type<- #name) }
    rule Parameter2Parameter {
        from s:owls!Parameter, s1: owls!serviceCategory
        to t : gcsd!Parameter (sampleValues<-'a parameter'),
           t1:gcsd!TypeReference (unitSymbol<- s.ParameterType, classID <-s1.categoryName, classificationSystemID<-s1.taxonomy),
           t2:gcsd!Description (value<- s.ParameterName, scope <- 'ParameterName', type<- #name) }
    rule  precondition2precondition {
        from s: owls!Condition, s1: owls!Expression
        to t: gcsd!Condition (conditionExpression<- s.expr),
           t1: gcsd!Expression (value <-s1.expressionBody, languageID <- s1.expressionLanguage),
           t2:gcsd!Description (value<- s.conditionID, scope <- 'preCondition', type<- #name) }
    rule result2postcondition {
        from s: owls!Result
        to  t: gcsd!Condition,
            t2:gcsd!Description (value<- s.resultID, scope <- 'postCondition', type<- #name) }
    rule grounding2Interface {
        from s:  owls!WSDLGrounding
        to t:gcsd!Interface (implementationTypeId<-s.wsdlDocument) }
    rule theServer2provider {
        from  s: owls!Participant (s.oclIsTypeOf(owls!TheServer) )
        to  t: gcsd!Provider,
            t1:gcsd!Description (value<- s.participantId, scope <- 'TheProviderId', type<- #name) }
    rule CompositeProcess2Phase{
        from
        s: owls!CompositeProcess(s.oclIsTypeOf(owls!CompositeProcess))
        to  t: gcsd!Phase ,
            t1:gcsd!Description (value<- s.name, scope <- 'phaseName', type <- #name) }
```

Figure 9: OWL-S transformation rules.

```
<?xml version="1.0" encoding="UTF-8"?>
<service:Service rdf:ID="congoBuyService">
    <service:presents rdf:resource="&congo profile;#CongoBuyProfile"/>
    <service:describedBy rdf:resource="&congo process;#congoBuyAtomicProcess"/>
    <service:supports rdf:resource="&congo grounding;#CongoBuyServiceGrounding"/>
</service:Service>
...
<profile:has_process rdf:resource="&congo process;#congoBuyAtomicProcess"/> "
<profile:serviceName> CongoBuyProfile</profile:serviceName>
<profile:contactInformation> www.CongoBuy.com </profile:contactInformation>
<profile:servicecategory categoryName="e-commerce" taxonomy="ecommerceTaxonomy"> </profile:servicecategory>
...
<process:AtomicProcess rdf:ID="CongoBuyProcess">
    <process:hasInput> <process:Input rdf:ID="creditCardNumber">
         <process:parameterType rdf:datatype="&xsd;#integer">creditCardNumber</process:parameterType>
    </process:Input> </process:hasInput>
    <process:hasOutput> <process:Output rdf:ID="purchase-receipt">
        <process:parameterType rdf:datatype="&xsd;#string"> purchase-receipt</process:parameterType>
    </process:Output> </process:hasOutput>
    <process:hasPrecondition> <expr:SWRL-Condition rdf:ID="creditLimit">
        <expr:expressionLanguage rdf:resource="#KIF"/>
        <expr:expressionBody rdf:parseType="Literal"> credit-limit>= ?purshasePrice </expr:expressionBody>
    </expr:SWRL-Condition> </process: hasPrecondition>
    <process:hasResult>
        <process:Result rdf:ID="newSolde"> </process:Result>
    </process:hasResult>
    <process:hasParticipant rdf:ID="try:TheClient">
        <hasParticipant:participantId="bookBuyer"/>
    </process:hasParticipant>
</process:AtomicProcess>
...
<grounding:WsdlAtomicProcessGrounding rdf:ID="CongoBuyServiceGrounding">
    <grounding:owlsProcess rdf:resource="#CongoBuyGrounding"/>
    <grounding:wsdlDocument rdf:datatype="&xsd;#anyURI"> http://schemas.xmlsoap.org/wsdl</grounding:wsdlDocument>
</<grounding:WsdlAtomicProcessGrounding
```

Figure 10: CongoBuy Bookselling service description in OWL-S.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:gcsd="www.gcsd.com" xmlns:try="http://www.example.org/try">
  <gcsd:GCSDservice name="congoBuyService"/>
  <gcsd:Service>
    <capabilities xsi:type="try:AtomicProcess" name="congoBuyAtomicProcess">
      <describes href="serviceInst.xmi#/"/>
    </capabilities>
  </gcsd:Service>
  <gcsd:ContactProfile/>
  <gcsd:Description value="www.CongoBuy.com" scope="contact information"/>
  <gcsd:Interaction involvedRoles="/7/@roleDescription.0"/>
  <gcsd:Function>
    <inputs xsi:type="try:Parameter" ParameterType="integer" ParameterName="creditCardNumber"/>
    <outputs xsi:type="try:Parameter" ParameterType="string" ParameterName="purchase-receipt"/>
    <preconditions xsi:type="try:Condition" conditionID="creditLimit">
      <expr expressionBody="credit-limit>= ?purshasePrice" expressionLanguage="#KIF"/>
    </preconditions>
    <postconditions xsi:type="try:Result" resultID="newSolde"/>
  </gcsd:Function>
  <gcsd:Description value="congoBuyAtomicProcess" scope="interactionName"/>
  <gcsd:Consumer>
    <roleDescription xsi:type="try:TheClient" participantId="bookBuyer"/>
  </gcsd:Consumer>
  <gcsd:Description value="bookBuyer" scope="TheConsumerId"/>
  <gcsd:Parameter sampleValues="a parameter"/>
  <gcsd:TypeReference classificationSystemID="ecommerceTaxonomy" classID="e-commerce" unitSymbol="integer"/>
  <gcsd:Description value="creditCardNumber" scope="ParameterName"/>
  <gcsd:Parameter sampleValues="a parameter"/>
  <gcsd:TypeReference classificationSystemID="ecommerceTaxonomy" classID="e-commerce" unitSymbol="string"/>
  <gcsd:Description value="purchase-receipt" scope="ParameterName"/>
  <gcsd:Condition/>
  <gcsd:Expression value="credit-limit>= ?purshasePrice" languageID="#KIF"/>
  <gcsd:Description value="creditLimit" scope="preCondition"/>
  <gcsd:Interface implementationTypeId="http://schemas.xmlsoap.org/wsdl"/>
</xmi:XMI>
```

Figure 11: CongoBuy Bookselling service description in GCSD.

4.3 Results Discussion

The whole process of implementing and testing our proposed transformations proves the correctness of our research hypothesis. Indeed, the heterogeneity of service descriptions hinders the interoperability. Offering services described using different, proprietary specifications promotes the vendor lock-in issue (Opara-Martins et al., 2014). Furthermore, the preliminary results that we obtained from transforming WSDL to GCSD and OWL-S to GCSD are an evidence that the unification of service descriptions enables their interoperability, combination, customization, and composition (Nguyen et al., 2012).

5 STATIC VALIDATION OF ATL TRANSFORMATIONS

Regardless of EMF being the de-facto standard modeling framework in the industry, it is still faced with a lack of an official validation and verification tool for EMF models with OCL constraints (González et al., 2012). Only two tools were found in the literature: (i) EMFtoCSP (González et al., 2012), which translates the model and its constraints into a constraint satisfaction problem, which is then solved by a constraint solver. EMFtoCSP checks the strong satisfiability, the weak satisfiability, the lack of constraint subsumptions and the lack of constraint redundancies in a model. However, the tool, which was developed as a research project, was not updated for several years. Therefore, the tool stopped from running at times, or it generated ambiguous results at other times; and (ii) EFinder (Cuadrado and Gogolla, 2020), which is a model finding tool that automatically searches for models satisfying a set of the model's OCL constraints. The approach enables EMF meta-model verification, verification of model transformations, and model synthesis. However, we did not get any results from it. EFinder was originally running over AnATLyzer (Cuadrado et al., 2016), which is a transformation validation tool for the static analysis of ATL model transformation. AnATLyzer is integrated with ATL environment, and it is available as an Eclipse plug-in, which enables the detection of typing and rule errors, as well as a support for pre-conditions and post-conditions. Therefore, we used AnATLyzer to correct errors (such as unresolved bindings, uninitialized features, rule conflicts, and others) in our transformations.

6 RELATED WORK

MDE-based approaches were proposed as solutions to achieve the cloud service interoperability. In (Alipour and Liu, 2018) the authors applied an MDE technique to manage auto-scaling services. Their approach enables the migration, deployment and configuration of services on multiple clouds. Ferry et al. proposed CloudMF *(cloud modeling framework)* (Ferry et al., 2013a), and they extended it in (Ferry et al., 2014) with: (i) a provider-agnostic management solution for applications that are deployed on IaaS and PaaS; (ii) an eclipse-based editor for the textual syntax alongside a web-based editor for the graphical syntax; and (iii) remote access to models@run-time reasoning engines. Furthermore, the authors proposed a domain-specific modeling language (DSML) called (CloudML) (Ferry et al., 2013b). Uni4Cloud was proposed in (Sampaio and Mendonca, 2011) to automatically configure and deploy applications across multiple clouds in an IaaS provider-agnostic manner. Whereas, MODAClouds (Ardagna et al., 2012) was proposed for service migration between clouds, and it provides a cloud-agnostic software design, a decision system to determine the most suitable cloud to deploy a given component and a support for migrating legacy software to the cloud. The approach in (Alipour and Liu, 2018) is different from our approach because the authors applied the transformation to transform a CPIM *(Cloud Platform Independent Model)* to a CPSM *(Cloud Specific Platform Model)* using EMF model generation abilities. In (Ferry et al., 2013a) (Sampaio and Mendonca, 2011), the authors adopted MDE techniques to enable the management of deployment, configuration and provisioning of PaaS and IaaS resources in multicloud. Whereas, in (Ardagna et al., 2012), the authors used MDE to enable the design and execution of applications on multicloud. Therefore, none of the aforementioned approaches applied MDE M2M technique to transform an input model to an output model based on a set of transformation rules and constraints.

7 CONCLUSION

This paper presents CIPiMo, which is a MaaS based on MDE to enable interoperability between cloud services described using heterogeneous CSD languages. The model enables the interoperability between services of different cloud models and for different target actors. The preliminary results that we obtained from our use case scenarios are promising, and they reveal that our model is capable of standardizing the descriptions of heterogeneous cloud services. Therefore, it enables their interoperability. As for future work, we will extend our model by defining the transformations for other CSD languages. Eventually, using a cloud simulation tool (for example: CloudSim) we will test the performance criteria of our MaaS model.

REFERENCES

Alipour, H. and Liu, Y. (2018). Model Driven Deployment of Auto-Scaling Services on Multiple Clouds. In *IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 93–96, Seattle, WA, USA.

Andrikopoulos, V., Reuter, A., Xiu, M., and Leymann, F. (2014). Design support for cost-efficient application distribution in the cloud. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 697–704, Anchorage, AK, USA. IEEE.

Ardagna, D., Nitto, E. D., Casale, G., and Petcu, D. (2012). MODAClouds: A Model-driven Approach for the Design and Execution of Applications on Multiple Clouds. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering*, pages 50–56, Zurich, Switzerland.

Barros, A. and Oberle, D. (2012). Handbook of service description. *USDL and Its Methods*.

Baudoin, C., Dekel, E., and Edwards, M. (2014). Interoperability and portability for cloud computing: a guide. *Cloud Standards Customer Council*, 1(1):1–31.

Bergmayr, A., Castilla, J. T., Neubauer, P., Wimmer, M., and Kappel, G. (2014). Uml-based cloud application modeling with libraries, profiles, and templates. In *CloudMDE 2014: 2nd International Workshop on Model-Driven Engineering on and for the Cloud co-located with the 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014)*, pages 56–65, Valencia, Spain. CEUR-WS.

Bezivin, J., Bruneliere, H., Jouault, F., and Kurtev, I. (2005). Model engineering support for tool interoperability. In *Workshop in Software Model Engineering (WiSME'2005)*, Montego Bay, Jamaica.

Boukhari, I., Bellatreche, L., and Jean, S. (2012). An ontological pivot model to interoperate heterogeneous user requirements. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, page 344–358, Heraklion, Crete, Greece.

Bouzerzour, N. E. H., Ghazouani, S., and Slimani, Y. (2020a). Cloud interoperability based on a generic cloud service description: Mapping owl-s to the gcsd. In *29th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2020)*, Basque Coast - Bayonne, France.

Bouzerzour, N. E. H., Ghazouani, S., and Slimani, Y. (2020b). A survey on the service interoperability in cloud computing: Client-centric and provider-centric perspectives. *Software: Practice and Experience*, 50(7):1025–1060.

Brambilla, M., Cabot, J., Wimmer, M., and Baresi, L. (2017). Morgan & Claypool Publishers, 2 edition.

Bruneliere, H., Cabot, J., and Jouault, F. (2010). Combining model-driven engineering and cloud computing. In *Modeling, Design, and Analysis for the Service Cloud-MDA4ServiceCloud'10: Workshop's 4th edition*, Paris, France.

Cabot, J. and Gogolla, M. (2012). Object constraint language (ocl): a definitive guide. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 58–90, Bertinoro, Italy.

Cheron, A., Bourcier, J., Barais, O., and Michel, A. (2019). Comparison matrices of semantic restful apis technologies. In *International Conference on Web Engineering*, pages 425–440, Daejeon, Korea. Springer.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web services description language (wsdl) 1.1. *W3C note*.

Cuadrado, J. S. and Gogolla, M. (2020). Model finding in the emf ecosystem. *Journal of Object Technology*, 19(2).

Cuadrado, J. S., Guerra, E., and de Lara, J. (2016). Static analysis of model transformations. *IEEE Transactions on Software Engineering*, 43(9):868–897.

Ferry, N., Chauvel, F., Rossini, A., and Morin, B. (2013a). Managing Multi-cloud Systems with CloudMF. In *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*, pages 38–45, Oslo, Norway.

Ferry, N., Rossini, A., Chauvel, F., and Morin, B. (2013b). Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. In *IEEE Sixth International Conference on Cloud Computing*, pages 887–894, Santa Clara, CA, USA.

Ferry, N., Song, H., Rossini, A., and Chauvel, F. (2014). CloudMF: Applying MDE to Tame the Complexity of Managing Multi-cloud Applications. In *IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 269–277, London, United Kingdom.

Ghazouani, S. and Slimani, Y. (2017). Towards a standardized cloud service description based on usdl. *Journal of Systems and Software*, 132:1–20.

González, C. A., Büttner, F., Clarisó, R., and Cabot, J. (2012). Emftocsp: A tool for the lightweight verification of emf models. In *First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, pages 44–50, zurich, switzerland.

Goscinski, A. and Brock, M. (2010). Toward dynamic and attribute based publication, discovery and selection for cloud computing. *Future generation computer systems*, 26(7):947–970.

Guana, V. (2015). Atlauncher. https://github.com/guana/ATLauncher.

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). Atl: A model transformation tool. *Science of computer programming*, 72(1-2):31–39.

Karim, R., Ding, C., Miri, A., and Liu, X. (2014). End-to-end qos mapping and aggregation for selecting cloud services. In *2014 International Conference on Collaboration Technologies and Systems (CTS)*, pages 515–522, Minneapolis, Minnesota, USA. IEEE.

Liu, F., Tong, J., Mao, J., and Bohn., R. (2011). Nist cloud computing reference architecture. *NIST special publication*.

Martin, D., Burstein, M., Hobbs, J., and Lassila, O. (2004). Owl-s: Semantic markup for web services. *W3C member submission*, 22(4).

Martino, B. D., Cretella, G., Esposito, A., and Sperandeo, R. G. (2014). Semantic representation of cloud services: a case study for microsoft windows azure. In *2014 International Conference on Intelligent Networking and Collaborative Systems*, pages 647–652, Salerno, Italy. IEEE.

Nguyen, D., Lelli, F., Papazoglou, M., and Heuvel, H. V. D. (2012). Blueprinting approach in support of cloud computing. *Future Internet*, 4(1):322–346.

Opara-Martins, J., Sahandi, R., and Tian, F. (2014). Critical review of vendor lock-in and its impact on adoption of cloud computing. In *International Conference on Information Society (i-Society 2014)*, pages 92–97, London, UK.

Perez, A. N. and Rumpe, B. (2014). Modeling cloud architectures as interactive systems. *arXiv preprint arXiv:1408.5705*.

Sampaio, A. and Mendonca, N. (2011). Uni4cloud: An Approach Based on Open Standards for Deployment and Management of Multi-cloud Applications. In *Proceedings of the 2Nd International Workshop on Software Engineering for Cloud Computing*, pages 15–21, Waikiki, Honolulu, HI, USA.

Zhou, J., Abdullah, N. A., and Shi, Z. (2011). A hybrid p2p approach to service discovery in the cloud. *International Journal of Information Technology and Computer Science*, 3(1):1–9.