A Comparative Study of Metamodel Integration and Interoperability in UML and Web Services

A. Staikopoulos and B. Bordbar

School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK
{B.Bordbar, A.Staikopoulos}@cs.bham.ac.uk

Abstract. The application of MDA to Web services has recently received considerable attention. Similar to UML diagrams, Web services are specialised languages each one targeting a specific aspect and functionality of the system. By using multiple languages, it is possible to specify complete integrated models of the system, having structure, behaviour, communication and coordination mechanisms. To benefit from MDA, Web service languages have to be represented as UML metamodels. In order to provide an overall view of the design and inter-operations of the system with models, it is crucial to integrate their UML metamodels. In this paper, we shall conduct a comparative study of the metamodel integration in Web services and UML. Drawing on the lesson learnt from the integration of Web services, a method of integration of UML metamodels will be presented, which facilitates model transformations and supports interoperability, inter-navigability and consistency across the integrated domains.

1 Introduction

The Model Driven Architecture (MDA) [9] is an emergent technology for software development promoting the automatic creation of models and code, by model transformations. The MDA aims to promote the role of models [7, 15], which are abstractions of the physical system emphasizing particular qualities for a certain purpose and are designed in a UML language or dialect. The model driven approach is based on the metamodel foundation to: a) specify the syntax and semantics of models and b) define the MDA mappings between source and target metamodels [12, 13].

Web services are self-contained, modular software applications that have open, standard based, Internet-oriented interfaces [1]. In a bigger scale, they can be considered as a set of interoperable technologies and standards, designed to support the integration of several autonomous and heterogeneous systems. Web services are particularly geared towards the Service Oriented Architecture (SOA) and paradigm [1, 16], which can be considered as a collection of services, coordinating and communicating with each other to support a specific functionality or concept of a system. A minimalist Web service architecture will contain the SOAP as the communication protocol, the WSDL to describe service interfaces, the UDDI as a service registry and the BPEL to specify executable processes for composite services [5]. If we consider

them within an overall architecture, for example SOA, they can specify service-integrated systems, having structure, behaviour, communication and coordination mechanisms.

The application of MDA to Web services is of major importance facilitating their design and development via automation. Recently, designing and developing Web services by MDA have received considerable attention [2, 3, 4, 11]. Despite the fact that Web service models are inherently integrated and created from multiple standards (languages represented as metamodels), their transformations are still considered in isolation and not as part of a whole mechanism. To explain, a business process expressed as a BPEL model has certain dependencies upon service interfaces modelled as WSDL models. This issue should be taken into consideration in the design and transformation of the process. Similar or even more complicated is the case when two processes communicate and exchange data, based on different formalisms such as BPEL and WSCI [1, 5]. Currently, such Web service languages when represented by metamodels appear to be unrelated from each other. In order to obtain a thorough view of the design and the inter-operations of our Web service models, we need to integrate their metamodels.

In this paper, we are examining modelling interoperable Web service systems and architectures, based on different but integrated metamodels. Specifying and formalising their model inter-relations and communication mechanisms is a very important issue for Web services, as they need to collaborate with each other to achieve their common targets. Thus, their capabilities are the result of integration and interoperability. As a result, the integrated metamodelling reflects their real accumulating characteristics and capabilities.

In order, to aid understanding, we need to clarify the terms of metamodel integration and interoperability, used throughout the paper. The descriptions given below are based upon well-established concepts in the field of computing.

- ***Integration****:* The creation of links between previously separated computer systems, applications, services or processes.
- ***Interoperability***: The ability to exchange. We need both to a) establish the mechanism to exchange (such as flow of information) and b) define how to extract or understand the information in order to process them.

Finally, the study is broken down into the following sections: Section 2 elucidates the shortcomings of creating isolated metamodels and highlights the advantages of providing integration and interoperability among the UML metamodels. Section 3 investigates the UML and Web service mechanisms, supporting integration, composition and interoperability for their models and schemas respectively. Section 4 compares and analyses them in order to assess their capabilities. Drawing on the lessons learnt, Section 5 presents a method of integrating Web service metamodels, with examples inspired from the Web service domain and their binding mechanism. Section 6 provides an overview of other approaches and various discussion points. Finally, Section 7 presents the conclusions made and summarises the benefits of the approach adopted by the authors.

2 Importance and Benefits

There are cases, where a domain (a subject area, having peculiar set of problems and concepts) [6] such as the Web service domain, may be composed from a number of other sub-domains for example BPEL (process-A), WSCI (process-B), WSDL (description) and SOAP (messaging) sub-domains. As a result, the Web service domain provides the composite or integrated metamodel, describing the overall domain and architecture of a potential system. Obtaining a thorough view of a system, by relating its inter-components is one reason for integrating the metamodels.

Moreover, the (sub) domains have to be interoperable, both horizontally such as processes with processes and vertically such as processes with descriptions and messaging, in order to function and interoperate correctly. Thus, in the first case we have to ensure the definition of meaningful sequence of actions and in the latter to use comprehensive communicative, messaging and transport mechanisms. This is the second reason to equip our metamodels with interoperability mechanisms.

In terms of development, integration and interoperability allows MDA to perform upon a combination of models, with well-established inter-relations and defined interactions. As a result the transformations are equipped with more detailed rules and mappings and generate better-linked artefacts both in terms of specific models and code.

In general, integration allows obtaining a thorough view of the system and its interoperations (assist design) and relating its internal components with established links (provide consistency and formalisation), while interoperability makes our integrated models more operational.

3 Integration and Interoperability Mechanisms in UML and Web Services

Web services have specific mechanisms for supporting the integration and interoperability of their XML artefacts. Similarly, UML has its own specified mechanisms for UML models. As our aim is to apply integration and interoperability among Web service metamodels as precisely as possible and closer to their physical implementation, we have to investigate and compare their mechanisms. In that way, we can apply their original XML integration mechanism to UML Web service metamodels. That will allow us to generate models semantically close to their original characteristics, as the domain metamodels will be equipped with their equivalent XML concepts.

The Object Management Group (OMG) and the World Wide Web Consortium (W3C) are two distinct organisations operating in two different domains, the modelling and specification of software systems and the Web standardisation and interoperability respectively. The OMG uses the UML family of languages and standards for software design and development, which are based on graphical models. UML languages are specified and defined upon a common core language, the Meta Object Facility (MOF) [12] providing the fundamental building blocks to construct and store metamodels. From the other side, the W3C uses the XML family of languages that are

textual specifications to specify Web services and standards. The XML languages are defined upon the XML Schema Definition (XSD) [19] to describe and constraint the structure and content of XML documents.

Both specification mechanisms (MOF in UML and XSD in XML) are comparable and can been seen as meta-languages, languages to define and create other languages. Thus, XSD is used to define WSDL and BPEL, and MOF to define UML and CWM [12]. As they are created from a common metamodel (XSD), the service languages conform to and share common semantics, so it is much easier to be integrated and be interoperable. Meta-languages play a very important role, as they define core domain characteristics that can be reused to define and create a variety of other metamodels, belonging to the same family of languages [13].

The UML specification is defined using the metamodelling approach and mechanism. The typical role of metamodelling is to define the semantics of how the model elements in a model get defined and instantiated. Web services on the other hand are defined by using a very flexible mechanism, the XML Schema, providing a way to specify the structure of XML documents.

Finally, the UML specification is organised into Infrastructure and Superstructure architectures. As a result, the various constructs defined are highly reused and coupled within the overall UML architecture. On the other hand, Web services can be seen as more independent entities that are part of more flexible or less coupled architectures. They can be integrated and combined in various ways with the flexibility of a component. An example of such architecture is the SOA [16], which is organised in separate layers regarding the functionality and concepts provided. In that sense the architecture can be seen as an accumulation of aspects.

Following, we identify, describe and compare the mechanisms defined within the UML and Web Service specifications supporting the fundamental ideas of composition, inter-relation, communication and extensibility of meta- elements.

3.1 Mechanisms for the Integration of UML Metamodels

The UML specification is defined in a number of metamodels and organised within hierarchical packages. The metamodels are gradually built upon more abstract modelling concepts, defined within fundamental reusable packages, such as the Infrastructure Library [13] and the Superstructure Kernel [14]. The UML language is organised in a four-layer metamodel architecture, separating the instantiation concerns across different layers, for example MOF from UML. According to that principle, the instantiation of meta-classes is carried out through MOF. The UML architecture has been designed to satisfy the following criteria [13].

1. **Modularity**: Group constructs into packages by providing strong cohesion and loose coupling.
2. **Layering**: Support a package structure to separate meta-language constructs and separate concerns (regarding instantiation).
3. **Partitioning**: Organise conceptual areas within the same layer.
4. **Extensibility**: Can be extended in various ways.

5. **Reuse**: UML metamodel elements are based upon a flexible metamodel library that is been reused.

To realise the above qualities, the UML specification is organised into two parts: Firstly, the UML Infrastructure defining the foundation language constructs where both M2 (UML) and M3 (MOF) meta-levels of the four-layer metamodel architecture are being reused. Secondly, the UML Superstructure extends and customises the Infrastructure to define additional and more specialised elements making up the modelling notions of UML.

The entire UML specification can provide an example of how metamodels representing different concepts, such as structure by a class or a component diagram and behaviour by an activity or an interaction diagram are integrated and connected together. In that case, even if they are defined within different packages they are interrelated and share common basic elements with other metamodels.

3.1.1 The UML Infrastructure

The Infrastructure Library [13] provides the basic concepts for organising and reusing metamodel elements. In the case of metamodel relation and composition, these are as follows:

Model elements to group elements together such as a) *Visibilities*, provide basic constructs for visibility semantics b) *Namespace*, provides concepts for defining and identifying a model element within a namespace c) *Package*, groups elements and provides a namespace for the grouped elements.

Model elements to specify some kind of relationship between elements such as a) *Generalisation*, specifies a taxonomic relationship between a more general and a more specific classifier b) *Redefinition*, specifies a general capability of redefining a model element c) *Association*, both a relationship and a classifier. Specifies a semantic relationship between typed instances d) *Element Import*, a relationship that identifies an element in another package, allowing the element to be referenced using its name without a qualifier.

Model elements defining basic mechanisms for merging their content such as a) *Package Merge*, specifies how one package extends another package, by merging their contents through specialisation and redefinition and b) *PackageImport*, a relationship that allows the use of unqualified names to refer to package members from other namespaces.

3.1.2 The UML Superstructure

The UML Superstructure [14] relies on the essential concepts defined in the Infrastructure to build the UML 2.0 diagrams. It provides a clear example of metamodel integration, as it brings together different packages from the Infrastructure library, by using package imports and merges. It redefines some of the concepts and further extends their capabilities [13].

The Superstructure in order to support the concepts of integration and interoperability among metamodels provides *additional relationships* [14] and links among elements. Examples of such links and relationships are a) *Dependency*, a relationship that signifies that a model element requires other model elements for their specification or implementation b) *Abstraction*, a relationship that relates two elements or sets of elements, representing the same concept at different levels of abstraction c) *Usage*, when one element requires another element (or a set) for its full implementation or operation d) *Permission*, signifies granting of access rights from the supplier to a client model element e) *Realisation*, a specialised relationship between two sets of model elements, one specifies the source (supplier) and the other implements the target (client) f) *Substitution*, a relationship between two classifiers, implying that instances can be at runtime substitutable, where instances of the contract classifier are expected g) *Implementation*, a relationship between a classifier and an interface signifying that the realising classifier conforms to the contract specified by the interface.

In addition, it defines a number of *composite structures* [14], providing more complicated elements with advanced capabilities. Examples of such elements are a) *Components*, representing a modular part of a system, which is replaceable within its environment b) *Connector*, a link enabling communication between instances. It can be something as simple as a pointer or something as complex as a network connection c) *Composite Structures*, a composition of interconnected elements, representing runtime instances collaborating over communications links to achieve some common objectives d) *Ports*, a structural feature of a classifier specifying a distinct interaction point between that classifier and its environment e) *InvocationAction*, invoke behavioural features on ports from where the invocation requests are routed onwards on links, deriving from attached connectors f) *Collaborations*, a kind of classifier defining a set of cooperating entities to be played by instances (its roles) and a set of connectors defining communication paths between the participating instances g) *CommunicationPath*, an association between two nodes, enabling them to exchange signals and messages.

3.1.3 Extensibility - Provide User Defined Elements

The UML specification is flexible supporting two types of extensions [7, 9, 12]. The first one is based on profiles and is referred to as a lightweight built-in mechanism. It does not allow the modification of existing metamodels but rather their adaptation with constructs that are specific to a particular domain, platform, or method. The second approach uses metamodelling techniques by explicitly defining new metamodel elements from pre-existing metamodels like the ones defined in MOF or the UML Infrastructure. This approach allows ultimate extensibility as it enables the definition of new concepts with new capabilities that can be tailored to represent precisely a particular domain of interest. There are various examples of extending UML, either with metamodels or profiles such as EDOC and the EJB Profile respectively.

3.2 Mechanisms for the Integration of Web service Standards

The Web service standards are built-upon the XML and XML Schema. As a result many of their characteristics and capabilities, such as extensibility and referencing, are based upon their defined concepts. By design, almost all Web service standards (as loosely coupled) [1] are designed to accommodate their integration and interoperability aspects flexibly. So, they define various points of extensibility, allowing them to interoperate with other standards and usually are separated into *abstract* and *concrete* parts.

The Web service implementations rely on the collaboration of various specifications to make them really functional and interoperable. For example, the SOA [16] is based on a collaboration of various Web service specifications, such as service description, discovery and messaging. Each one can represent either a particular layer (regarding the SOA architecture) or a particular functionality or concept. In this sense, they can be compared with UML modelling that is based on a combination of various metamodels, belonging to different packages, to make up a complete system specification.

An example of how Web service specifications can be combined and interoperated to fulfil an objective is a business process request defined as a BPEL *Invoke* operation [5]. The process defines the implementation logic of a service, accessed from specific interaction points via a set of Web interfaces defined in WSDL and sending a SOAP message representing a particular request to a business participant [1]. The participant replies back by triggering an equivalent mechanism, with an appropriate formatted XML message.

In this paper, we are interested in investigating these inter-relationships and dependencies among fundamental Web service standards such as BPEL, WSDL and SOAP and examine how they are combined to support service interoperability at different levels of abstraction and implementation, by realistic examples and cases. The following are defined elements and technologies used for the integration, extensibility, collaboration and communication of schemas and Web service specifications.

3.2.1 Common - Core Characteristics

The XSD mechanism [19] provides the core/fundamental characteristics of Web services, utilised by almost all Web service standards, allowing mechanisms for grouping, extensions and referencing. The XSD also provides the foundation mechanism for constructing messages and datatypes that are the means of interoperability and communication for services.

The elements supporting such mechanism are a) *Schema***,** is associated with a namespace and provides a grouping for defined XML elements b) *Namespace***,** provides identification and access rights for its elements c) *Import,* allows to use schema components across different namespaces with references d) *Include,* assembles schema components to a single target namespace from multiple schema definitions e) *Redefine,* allows the redefinition of one or more components f) *Redefinable,* specifies an element so that it can be redefined g) *SubstitutionGroup,* supports the substitution of

one named element for another h) *Extension*, provides the mechanism to extend the element content i) *Restriction*, provides the mechanism to restrict the element content j) *References*, provides pointers to already defined elements such as *GroupRef, AttributeGroupRef* and *ElementRef*. Finally, k) *AnyURI, AnyType and AnyAttribute* can point to any location, any type and extend an element with attributes not specified by the schema respectively.

3.2.2 Specialised Characteristics

More specialised characteristics and concepts are usually defined within each service specification. For example, the WSDL defines how Web service interfaces can be defined in terms of protocol bindings, port-types, operations and messages. Analogously, other specific application constructs have been defined for BPEL and SOAP [1, 11]. Those standards can be combined and integrated together by specifying a binding mechanism that actually provides the links among the involved service specifications. So, tools and engines can realise those implementations and execute the actual Web service collaborations across different standards and protocols.

Almost all Web service standards are designed to accommodate interoperability and be extensible. As a result, their specifications at certain points, particularly their concrete parts are defined in a way that leaves space for different implementations. It is very similar to UML or Java when a *Classifier* or an *Object* can be of any type, therefore the model or application can support different implementations or behaviours.

3.2.3 Auxiliary Characteristics

There are also various auxiliary technologies and standards that are used to provide more sophisticated mechanisms, such as for relating elements across multi-documents. Those can be easily used and embedded within specifications to create more elaborate structures and complicated functionalities. Such a mechanism is the *XPath* [18] language for addressing various parts of an XML document. It provides the means of linking elements together; therefore it can be represented by a relationship in UML modelling.

4 Analysis & Classification of Mechanisms

At this point, we should analyse and classify the integration and interoperability mechanisms supported by UML and Web services, upon the following fundamental criteria:

Structures: Provide the ability to implement a) *Containers*, group and identify model elements within collections and b) *Composite Structures*, provide the means and concepts to integrate/combine together different model elements into new or coupled entities.

Dynamics: *Messaging*, establish the concepts and mechanisms to specify dynamics for example perform invocations, interactions, messaging upon established connections.

Links: Provide *Relationships/Addressing* mechanisms to establish semantic relationships among model elements that may belong to different groups.

Mechanisms: Define *Mechanisms upon containers*, operate upon elements or collections of elements. New groups of elements may be generated as a result of intersection and union operations.

According to these criteria, the UML and Web service capabilities are compared, assessed and described as in the following tables:

Table 1. The UML mechanism supports criteria by specifying the following UML modelling elements

	UML support
Containers	Initially provided by the Infrastructure library and specialised from the Superstructure kernel. Examples are Namespace, Visibility and Package model elements
Comp. Structures	Various metamodel elements are defined such as components, composite structures, collaborations
Messaging	A number of actions are specified for messaging, invocations and their supporting concepts like input and output pins
Links	There are various types of semantic relationships such as permissions, substitution, generalisation, usage etc
Mechanisms	Package mechanisms, like package import and merge

Table 2. The Web service mechanism supports criteria by specifying the following XML elements

	Web service support
Containers	Schema, Namespace
Comp. Structures	ComplexType, Group, Sequence etc
Messaging	XML Datatypes, SOAP messages
Links	Element, Attribute and Group references, Ids
Mechanisms	Import, Include, Redefine

4.1 Compare & Contrast Mechanisms

It is clear that both XSD and MOF or the UML Infrastructure are meta-languages for the modelling and Web service domain respectively. As such, they provide the fundamental concepts and building blocks to define and create other languages such as the WSDL, the BPEL or the UML and the CWM. Regarding the four-layered metamodel architecture they both belong to M3 level [12]. In addition, they are both self describing and reflective as they have been specified by their own means and concepts.

The UML and Web service specifications, as seen in section 4, both define integration mechanisms, however they are designed in such a way to support and reflect the

characteristics of their domains. Therefore, UML is designed to create more compact models through meta-models. UML also seems more integrated as it is designed from one organisation all-over. Metamodels via diagrams can provide different views of the same system. In that case, their integration points are well defined and bound, as effectively they belong to the same specification. On the other hand, Web service standards are more loosely coupled and are developed rather independently from each other. They are integrated and combined later on into functional units, using previously defined, abstract integration points. In addition, their specifications are defined in textual XML Schemas.

To conclude, we can say that the UML specification is focused on separating the different views of a system by providing different diagrams that are somehow related together by their metamodels, while Web service specifications are concerned with logically separating the *abstract* from the *concrete* parts [1, 19].

The result of the comparison will justify and influence the approach adapted, of how to implement the integration and interoperability mechanism with its domain metamodel, in a way to reflect as accurately as possible their actual characteristics.

5 A Method to Support Metamodel Integration & Interoperability

Currently, the Web service domain is scattered into several different metamodels such as BPEL, WSDL and SOAP [1]. Our approach on supporting the integration and interoperability of metamodels is based on the concept of relating all these unconnected metamodels in a cooperating fashion, through specified and formalised links. As a result, we introduce the binding mechanism (please refer to Fig. 1) as a metamodel. The method is influenced from the Web service domain. However, we believe that the approach is similarly applicable to other domains, as it can be considered neutral and generic enough. The binding metamodel integrates the metamodels together in a formalised way, by specifying both their static aspects (by defining interrelationships) in order to provide integration and dynamic aspects such as communication, interactions and conformance to provide interoperability among the metamodels.

The steps to apply our approach can be described as follows:

Firstly, we need to identify the metamodels composing our system or domain of interest. One can either create these metamodels or reuse predefined ones. Secondly, the relationships, dependencies or interaction points among these metamodels have to be identified. Thirdly, the mechanisms supporting integration and interoperability between the actual *"real"* domain (in the case of Web services, by XML and XSD) and modelling domain (by metamodels) are compared and checked whether they are harmonised. This should allow to design their actual supported mechanisms (originally expressed in XSD) in another formalism (metamodel) as precisely as possible, providing clear domain models and encapsulating their original domain capabilities accurately. Having identified the supported mechanisms (for example how to relate to elements), the binding metamodel is introduced, where the relationships, properties and rules are applied accordingly. At this stage the integration points among the

metamodels are being defined. Following, upon the integration points (providing relationships and links) and within the binding metamodel, we further specify the communication mechanism in terms of interactions, messages and data-types exchanged, making our models functional and interoperable. Lastly, we may have to model the binding mechanism with equivalent UML models as precisely as possible in order to respect its internal semantics for example establish object roles within collaboration diagrams.

5.1 Applying the Binding Mechanism – Web Service Examples

Following, a number of examples from the domain of Web services are provided to illustrate how the binding metamodel can be applied among Web service metamodels. As already mentioned, the Web service domain can be represented by various metamodels. In these examples, we are going to consider two metamodels: the BPEL and WSDL, representing a partial view of the Web service domain. The metamodels can either be created from their original XSD specifications or reuse pre-existing ones [2, 3, 4].

Fig. 1 illustrates how the Web service domain is represented by the BPEL and WSDL metamodels and how these are mapped to equivalent UML modelling concepts, providing their platform independent representations, as UML activity and component diagrams respectively. Regarding these mappings, there are already some research activities defining their MDA transformations [2, 4]. On the right side, one can see how a metamodel binding between the BPEL and the WSDL metamodels is applied and how this is reflected as a UML representation (in this case a collaboration element). The UML model that would be selected needs to encapsulate the binding mechanism as semantically precise as possible and provide all the necessary concepts and capabilities for representing integrations and collaborations of its participating parts.

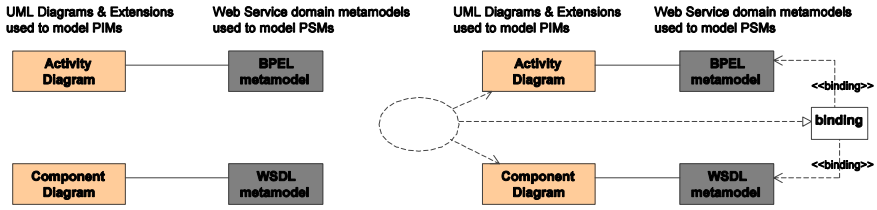

Fig. 1. The UML & Web service domains with equivalent mappings (on the left) and required bindings (on the right)

The binding metamodel in Fig. 2, represents the BPEL and WSDL integration. It defines two additional model elements, the *PartnerLinkType* and *Role* together with their inter-relationships with the other model elements from BPEL and WSDL. In this case, the binding metamodel is attached to the actual WSDL metamodel as an extension, meaning that both WSDL and binding model elements share the same namespace.

The services with which a business process interacts are modelled as *PartnerLinks* and are performed upon Web service interfaces. Each *PartnerLink* is characterised by a *PartnerLinkType* maintaining the conversational relationship between two services by defining *Roles* played by each of the services in the conversation 19. Each *Role* specifies exactly one WSDL port type. The relationship among partners is typically peer-to-peer (such as BPEL to BPEL) and can be modelled by a two-way dependency. The *PartnerLink* declaration specifies the static shape of the relationship that the process will employ in its behaviour.

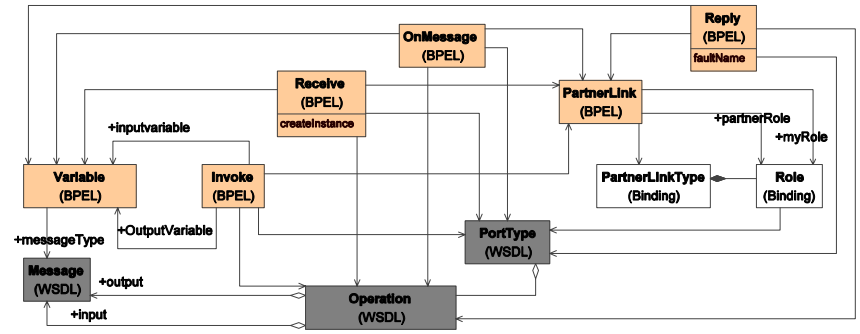

Fig. 2. A metamodel binding example between BPEL & WSDL supporting integration

Afterwards, the interaction mechanism of the binding metamodel is being specified, by identifying its dynamics in terms of establishing interactions, message exchanges in various patterns and use comprehensive data-types as illustrated in Fig. 3. That will permit our models to be interoperable across their integrated points, meaning that their instances can be really executable [17]. The integrated points defined as relationships among model elements specify the links where messages or signals can travel. Such formalisation provides consistence, as interactions can be performed only through these established points following a specific interaction pattern.

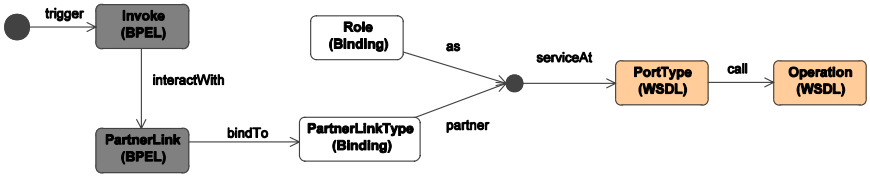

Fig. 3. How the metamodel binding supports interoperability

In particular, Web services by design support interoperability, as they are based on XSD for specifying datatypes, SOAP for messaging and communications and WSDL for describing interactions as exposed operations. Thus, a Web service example as in this case would not encounter severe inconsistency problems of that kind, as the interactions would be performed upon common standards. For example, in order to perform a simple invocation call across collaboration processes, the following sequence

will occur among involved metamodel elements to transmit a message across the integrated points and according to a specific interaction pattern (in this case without a reply):

BPEL1➔binding1➔WSDL1➔SOAP➔WSDL2➔binding2➔BPEL2 (**1**)

Following, Fig. 4 illustrates how the metamodel mechanism can be realised by a set of instances, participating in an invocation operation *Invoke* from a *shippingServiceCustomer* to another *shippingService* participant in order to handle the shipment of orders. The actual example is derived from the BPEL specification [5].

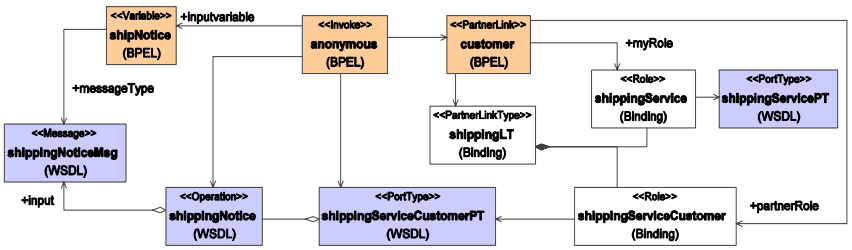

Fig. 4. Instance Examples upon binding

Finally, we examine how to represent the binding mechanism in a platform independent manner with an equivalent UML model. For that reason, one needs to utilise the UML 2.0 composite structures, such as *Collaboration* and *StructuredClasses* that can support the composition of interconnected elements, representing run time instances, collaborating over communication links to achieve common objectives [14]. In particular, *Collaborations* provide the means to define common interactions between objects and other classifiers and assign responsibilities in terms of roles. The interaction is similar to providing a pattern of communication between parts and is specified as a set of messages passed between objects playing predefined roles.

In this case, the UML 2.0 Collaboration modelling element is used to encapsulate the *PartnerLinks* playing various *Roles* by each of the services participating in the conversation and providing a semantically mapping to our domain binding metamodel.

Following, Fig. 5 depicts the binding mechanism as an independent model, explaining how the binding mechanism works, by describing the structure of collaborating elements (roles), performing specialised functions upon defined communication paths of participating instances. In that case the binding metamodel and mechanism are rather simple and can be represented by a UML *AssociationClass* [13]. That is to say a class that defines specific properties upon an established semantic relationship between classifiers, in this case the metamodel elements from BPEL and WSDL.

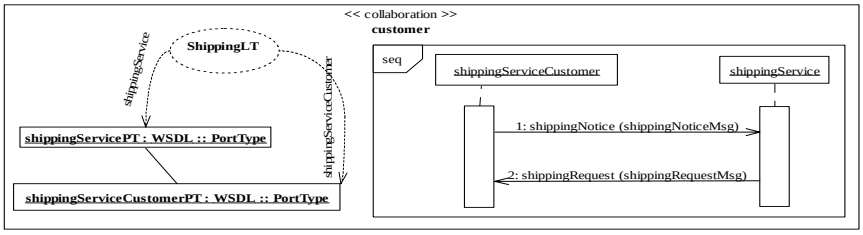

Fig. 5. UML collaboration, modelling bindings in UML

6 Related Work & Discussion

There are several approaches on metamodel composition and interaction [8, 10, 15]. More specifically, in [10] the key feature of this approach is the combination of new metamodels from existing metamodels, through the use of newly defined operators such as equivalence, implementation and interface inheritance. The approach emphasises the compositional operations for creating new concrete metamodels and not actually relating them as cooperating entities, as in our case. Next, the approach in [15] suggests the integration of metamodels by a joint action model. The approach is based on message interactions, thus supporting more dynamically and interoperable models. In that respect, the approach is comparable to our method on establishing interaction points across the metamodels. Finally, the approach in [8] is based on extending the UML language for the composition of domain metamodels by proposing a UML profile.

In our study our effort is to establish both connections upon the metamodel elements to support integrations in a cooperating way and communication mechanisms to support their interaction characteristics. In that way we can provide consistency and formalisation upon the interconnected metamodels, properties that are necessary for performing model transformations.

7 Conclusion

Integration and interoperability are very important issues for composite domains as in the case of Web services, as they need to collaborate with each other to achieve their common targets. Their emerged capabilities can be seen as the product of integration and interoperability. In modelling, Web service languages need to be represented by equivalent metamodels. By defining their metamodel relationships and interaction mechanisms, we support the integration and interoperability of their models. In that way we can design complete system models having formalised interaction points and perform model transformations across multiple connected metamodels.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V., Web Services Concepts, Architectures and Applications, Data-Centric Systems and Applications, ISBN: 3-540-44008-9, (2004)
2. Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F.: An Experiment in Mapping Web Services to Implementation Platforms. Technical report: 04.01, LINA, University of Nantes, Nantes, France (2004).
3. Bordbad, B., Staikopoulos, A.: Modeling and Transforming the Behavioural Aspects of Web Services. In: Proc. 3rd Workshop in Software Model Engineering - WiSME2004, UML (2004)
4. Bordbar, B., Staikopoulos, A.: On Behavioural Model Transformation in Web Services. In: Proc. Conceptual Modelling for Advanced Application Domain (eCOMO), Shanghai, China (2004), p. 667-678
5. BPEL: BEA, Microsoft, IBM, SAP, Siebel, Business Process Execution Language for Web Services, Version 1.1. (2003)
6. Greenfield, J, Keith Short: Software Factories, Wiley, ISBN: 0471202843, 2004
7. J. Siegel, Developing in OMG's Model Driven Architecture, Object Management Group, November (2002)
8. Jacky Estublier, A.D.I. Extending UML for Model Composition. in Australian Software Engineering Conference (ASWEC). 29 March, 1 April (2005). Brisbane, Australia.
9. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture-Practice and Promise. (2003)
10. Ledeczi A., G.N., G. Karsai, P. Volgyesi, M. Maroti. On Metamodel Composition. in IEEE CCA 2001. September 5, (2001). Mexico City, Mexico.
11. Lopes, D., Hammoudi, S.: Web Services in the Context of MDA. In: Proc. 2003 International Conference on Web Services (ICWS'03) (2003)
12. OMG: Meta Object Facility 2.0 Core Specification. (2003). Document id: ptc/03-10-04
13. OMG: UML 2.0 Infrastructure Specification. Document id: ptc/03-09-15 (2003)
14. OMG: UML 2.0 Superstructure Specification. Document id: ptc/03-08-02 (2003)
15. P. Denno, M.P.S., D. Libes, E.J. Barkmeyer, Model-Driven Integration Using Existing Models. IEEE Software, Sept./Oct. (2003): p. 59-63.
16. Rakesh Radhakrishnan, Mike Wookey, Model Driven Architecture Enabling Service Oriented Architectures, Sun Micro Systems, March (2004)
17. Stephen J. Mellor and Marc J. Balcer: Executable UML a Foundation for Model Driven Architecture. Addison Wesley. ISBN 0-201-74804-5, 2002
18. W3C, XML Path Language (XPath) 2.0, W3C Working Draft. July (2004)
19. XML Schema W3C, XML Schema Part 0: Primer, W3C Recommendation, May (2001)