

Experimental transformations between Business Process and SOA models

Akira Tanaka[†], and Osamu Takahashi[‡]

[†]view5 LLC, Japan

[‡]School of Systems Information Science, Future University Hakodate, Japan
a.tanaka@view5.co.jp, osamu@fun.ac.jp

Abstract - in designing enterprise IT systems, two major architectural styles exist today: process-oriented and service-oriented architectures. Either one of them can be used to define behavioral aspects of the business specifications. In reality, a process can make use of various services, and a business service can be implemented as a process. This duality applies to such technology as BPMN and SOA. In RM-ODP standard, however, both are part of a standard viewpoint language, and they complement with each other. In this paper, using a textual domain specific language and a tool supporting it to capture the essence of those modeling languages, we examine the relationship between process-based specifications and service-based specifications for a typical buy-sell-ship business process or collaboration. Architectural comparison is done by examining a model transformation of process to service, and service to process. The difference of the two types of model and the implication of the results are discussed.

Keywords: business process; service-oriented architecture; enterprise architecture; RM-ODP; model transformation.

1 INTRODUCTION

Enterprise architecture is widely used as a way to describe overall architecture of enterprise systems. There are many approaches to define enterprise architecture. For instance, to describe everything from hardware components to business strategies, a matrix of concerns and perspectives is used in Zachman Framework[1], or a set of customized architectures such as application architecture to capture different aspects is used in Federal Enterprise Architecture[2] initiatives. There are also requirements to harmonize businesses and ITs. In this context, the importance of business specifications, which describes “what to achieve” at business level, is being recognized.

There are a variety of approaches to define business specifications: business process oriented approach that uses “Business Process Model and Notation” (or BPMN[3]), business rule oriented approach that uses “Semantics of Business Vocabulary and Business Rules” or (SBVR[4]), business events based approach such as “Event Driven Architecture” (or EDA) and “Complex Event Processing” (or CEP[5]), business service oriented approach such as “Service Oriented Architecture” (or SOA[10]), and many more. Among them, two major architecture styles exist, which are process-oriented and service-oriented architectures. They are mainly used to define behavioral aspects of the business specifications.

According to BPMN specification, “business process is a defined set of business activities that represent the steps required to achieve a business objective.” This implies, when top down design approaches are taken, a business objective is set first, and a process or a set of processes is/are defined to achieve the objective. The IT system will be designed to provide necessary functionalities to the defined steps in the process. In this approach, interested behaviors or interactions exist between steps and among all the participants.

According to SOA Reference Model, “A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.” In this approach, most application elements will be implemented as services so that client software can find and consume necessary services to achieve its goal. The interactions in this model are between consumer and provider of the service. Orchestration of the services is not within the scope of this architecture.

Both styles are used as foundational architectures when developing enterprise IT systems. It should be noted, however, that a process can be decomposed into steps, each of which may consume services. And, a service can be implemented as a process. This duality applies to business systems designed based on current technologies such as “Unified Modeling Language”¹ (or UML[6]), BPMN and SOA.

The issue we have is “are they really different, or are they different sides of the same coin?” In other words, are they only different in architecture styles and equivalent in capabilities? And if so, how can we measure the equivalence? In this paper, we will examine this issue using modeling technique including UML, “Domain Specific Language” (or DSL[8, 9]), and model transformations. If they are essentially equivalent, there should be correspondences between them, that is, a service based model should be able to be transformed into a process based model, and a process based model should be able to be transformed into a service based model.

We will first look at business process oriented approach and examine how much it can be mapped to SOA approach. We will then look at service based approach and examine how much it can be mapped to process oriented models. In doing this, we will use DSL and model to text transformation tool.

¹ UML is a standard graphical modeling language for analyzing, designing and implementing software-based systems. The current version is UML 2.0.

2 BUSINESS PROCESSES

Business specifications are usually the most examined specification in Enterprise Architectures, since most business users, in addition to technology providers, would need to review it to see if it correctly captures the business requirements. When top down approach is taken, it usually starts with analysis of business environment, establish a new goal and strategies, design business processes to achieve the goals. There are multiple choices in diagramming business processes. UML Activity Diagram and Business Process Modeling and Notations (BPMN) are the most used ones. In this paper, we use ODP ([7] see IV) Process diagrams that is a slight extension to UML Activity Diagram, since it has necessary characteristics to do the experiment.

The diagram on the right side of this page (Figure 1) shows a sample purchasing business process among three parties, buyer, seller, and shipper in ODP Process Diagram. Each lane represents role and behavior of the party, and each step is represented as an Action internal or external to the lane. In case of external Action, the control flow crosses the lane with or without artifact passing. Artifact, represented as ObjectNode, is there to capture necessary business information to be passed. There are split/merge used to control, i.e. to create and conclude, parallel activities. This is almost the basic Activity Diagram except for applied ODP stereotypes. The dotted lines are there to show logical grouping of the steps that have certain meaning in the application, e.g. placing an order. Note that these dotted lines are not part of standard notation, and they should be read as additional comment. Although it is possible to group steps using sub-process, it does not provide improvement in readability of the process, and therefore we did not take that approach.

In summary, this process diagram shows participants of the purchasing process and a collection of necessary steps in a prescribed manner leading to the objective. IT systems will be designed to support some portions of the steps. This style is effective when an IT system is to be built against pre-defined business processes (i.e. what needs to be done in what order).

3 SERVICE OR BUSINESS SERVICE

The definition of term “Service” in SOA is still under open discussion. However, there is one in OASIS’s SOA Reference Model[10], which is “A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.” OMG’s SoaML[11], which is a UML Profile for Service oriented architecture Modeling Language, is a standard to describe SOA based models. A slight modification of above definition was used there, which says “A service is value delivered to another through a well-defined interface and available to a community (which may be the general public). A service results in work provided to one by another.” In this UML Profile, various SOA concepts are defined. For instance, a community is defined as a place for participants

consume/provide services to each other. For each pair of participants they have service contract that govern the behavior aspects when consuming/providing services. As for diagramming, it mainly uses UML Collaboration Diagram, Class Diagram and Component Diagram.

If compared with the previous process model, participant can be considered as role. The diagram on the right side of this page (Figure 2) shows a sample high level Service Architecture using SoaML to represent Buy-Sell-Ship collaboration.

Note that there are other kinds of diagrams associated with this high level description. For instance, Service Contract Diagram contains two roles, consumer and provider, with a sequence diagram specifying service message exchanges when provided services are consumed.

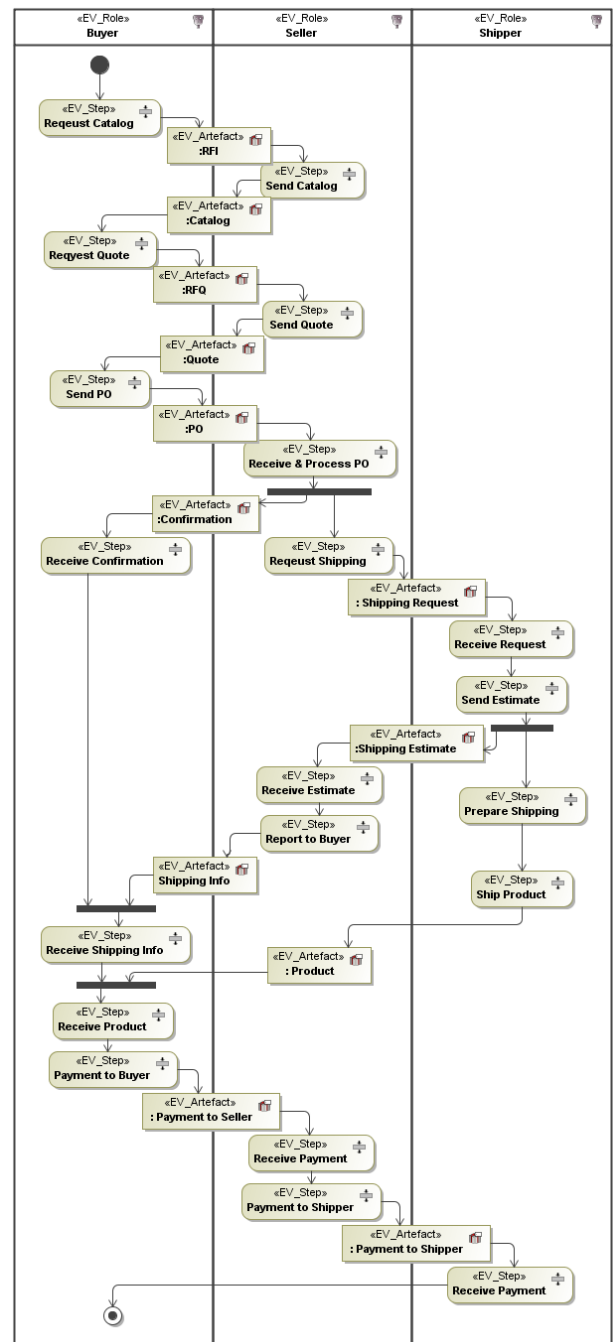


Figure 1: Sample ODP Purchasing Business Process

In summary, the service diagram (Figure 2) shows participants of the collaboration and service contracts, which includes interface and behavior definition, between service consumers and service providers. Note that there is no “steps taking place in a prescribed manner” defined. IT systems will be designed to support service providers and consumers. This style is effective when an IT system is to be built using existing or to be built services, such as newly developed internal services, wrapped legacy functions, or external services using web services, with a flexibility of clients’ choice of the services.

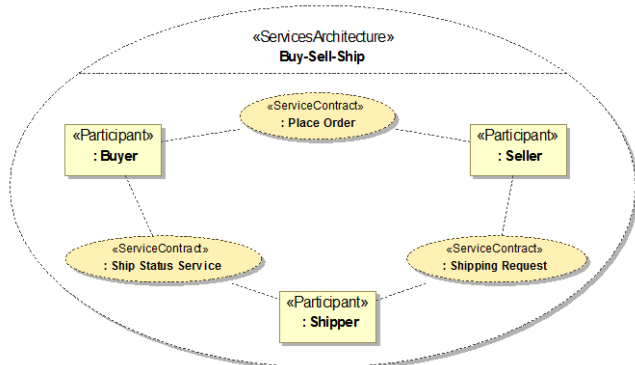


Figure 2: Sample Service Architecture

4 RM-ODP

RM-ODP stands for Reference Model for Open Distributed Processing, which is a family of international standards for developing standards for open distributed processing systems. It is a set of reference models, and it also has UML Profile standard to represent its concepts using UML tools. This standard is used as an open Enterprise Architecture, and we use this standard with associated UML Profile (e.g. in Fig. 1) to show something not biased to specific process modeling notations. It defines five standard viewpoints, but we will use or refer to only three of them in this paper: Enterprise, Information, and Computational viewpoints.

5 DOMAIN SPECIFIC LANGUAGE (DSL)

According to Domain-Specific Languages[8], domain – specific language is defined as “a computer programming language of limited expressiveness focused on a particular domain.” DSL could be graphical or textual, could be internal (designed based on general purpose language) or external (having no specific host language). In this paper, textual and external DSLs for process oriented modeling (ProcessDSL) and service oriented modeling (ServiceDSL) are described and used.

6 PROCESS TO SERVICE MODEL TRANSFORMATION

Model transformation achieves one source model described in one specific language to be converted into target model described in other specific language, without violating rules for those languages. Typical example is to transform UML Class Diagram to Relational Table. Four

layer meta architecture is usually used to explain the mechanism. As a standard, OMG’s MOF/QVT[12] is the best known one. As open source projects, widely known examples are eclipse ATL[13, 14] and QVT.

The next table shows metamodel and main elements of Process models and Service models related to our sample process.

Table 1: Metamodel Elements

	Business process	Service
Metamodel	Process related part of RM-ODP Enterprise Viewpoint metamodel	SoaML metamodel
Main Elements	Object, Role, Process, Step, Action, Activity, Artefact, Interaction	Participant, Service Contract, Service Architecture, Service Interface, Service Choreography, Message, UML Collaboration/Component

The following is a summary of how service metamodel element could be created with given process metamodel elements.

A. Participant/UML Component

A Participant (actually a Type) is equivalent to a Role in process. Since Participant deals with computation, a computational object with the same name should be introduced or assumed in the Process model side.

B. Service Contract

Service Contract can be considered as a concept representing interaction or behavior between two roles. ODP’s Interaction is the closest concept, but it is not really a part of process modeling. Service Contract uses Sequence Diagram to represent the behavioral aspects.

C. Service Architecture/UML Collaboration

Service Architecture’s closest concept is a collection of Interactions among all the involved roles.

D. Service Interface

The concepts like Interface, Operation, and Signature belong to Computational Object in ODP. If we could assume the existence of Computational Object with the same name as Participant, they are the corresponding elements.

E. Service Choreography

Service Choreography defines ordering of service messages between service consumer and provider. This can correspond to an ordered sequence of ObjectFlow involving the corresponding two roles in the process model.

F. Message

Information viewpoint of RM-ODP is the viewpoint where all the concerns on information within the system are defined. However, in Enterprise and Computational viewpoint, there is a need for information model and they are created based on the one defined in Information

viewpoint. The Message data types are a collection of data types and structure of data elements visible in Enterprise viewpoint, and therefore those should be prepared as a part of process model.

From above, we can observe that when transforming a process to a service, the process needs to be decomposed into an ordered set of two party interactions, and interaction should be brought into a process model.

Figure 3 is a sample ODP Interaction Diagram showing Buyer as an initiator of the interaction, Seller as a responder of the interaction, and the references to various artifacts that are actually electronic document. The behavior of both sides is described using UML State Machine, which is different from the case with Service Contract.

When considering the model transformation, this Interaction Diagram could act as an intermediate from process to service, meaning that we can transform a process model to interaction model, and then transform it to service model.

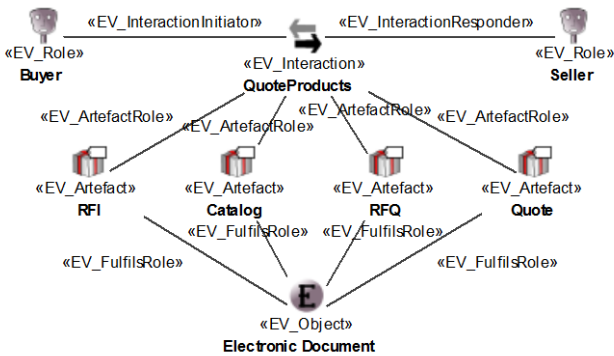


Figure 3: ODP Interaction Diagram

7 SERVICE TO PROCESS MODEL TRANSFORMATION

This section considers the reverse of the previous section, which is about transformation from service model to process model. Service model here is a model based on SoaML.

A. Object

SoaML's instance of Participant can be considered as Enterprise Object in a limited sense, since it also consumes and provides services like Computational Object. Message is a good source for defining Information Object.

B. Role

Participant is almost the equivalent to role. At the same time, Participant can be understood as Computational Object, which takes all the interface information from the Participant.

C. Process/Activity

Service Choreography specifies dynamic binary relationship of the behavior, and can be used to construct a portion of a Process. However, constructing whole process is not possible, since there is no orchestration information is available for construction of the whole process, including where to start the process.

D. Step/Action

Step/Action means action execution, which is defined in Service Contract and Service Choreography. However, it is

not possible to generate Steps/Actions which do not correspond to interactions such as internal actions.

E. Artefact

Message is the only element to map to Artefact.

F. Interaction

Service Contract corresponds to Interaction where Consumer side corresponds to Initiator side and Provider side corresponds to Responder side.

8 MODEL TRANSFORMATION METHOD USING TEXTUAL DSLS

When we refer to “model transformation,” it usually means transforming models created with UML tools or some other specific tool such as BPMN tools (graphical tools). Those models are actually saved as text file, for instance as a form of XMI[15] or XML, and then transformation logics are applied to it. However, UML itself is a complex specification and that is reflected in XMI. A simpler way to experiment some modeling issues without involving too much complexity was needed, and that was the reason we took DSL.

We will now explain a method of model transformation using textual DSL. With textual DSL such as the one developed with eclipse/Xtext[16], a grammar is first defined, and the grammar based model editor is generated so that user can create his/her own model based on the DSL.

Suppose you have two textual DSLs: ProcessDSL for process modeling and ServiceDSL for service modeling. If you define a grammar for ProcessDSL, you get the ProcessDSL editor. The same is true for ServiceDSL. Once a process model is defined, a template is applied to the model to generate output text (e.g. source code or XML file). Here, it is possible to design a template to generate text, which has a structure that ServiceDSL editor imposes. This is not always possible, since the source model may not contain necessary information to transform to. But, if it did, this model to text transformation works as a model to model transformation. This is the basic idea we used for Process to Service and Service to Process model transformations.

In order to achieve this, we have created above DSLs as simple textual DSLs to capture core concepts of ODP Process Diagram (or UML Activity Diagram) and SoaML Diagram respectively. They are simple, because not all concepts are used and some complex concepts were simplified to some extent in the grammar.

The tool used is eclipse/Xtext and its integrated model to text transformation engine Xpand/Xtend. But, with any other textual DSL tooling, such as Spoofox[17] or MPS[18], this can be done in the similar way.

Figure 4 shows a portion of the Process DSL grammar.

Using generated DSL editor, a process model in this ProcessDSL is created (Figure 5), which is done by typing, not by applying model to text transformation from the process model (Figure 1). The last step is to define a template to generate text, which is explained in the next page. The Xtext grammar files and sample model are published on the following web site[19].

We also created InteractionDSL based on ODP concept of Interaction, and a sample model is shown in Fig. 6. With

this InteractionDSL, multiple artifacts are allowed to be shown, but in our transformation rule, only one artifact per interaction is generated. This is because ODP interaction diagram shows aggregation of multiple interactions, but our process to service transformation is targeted to each crossing the lane interaction only.

```

interaction PlaceOrder {
    initiator Buyer
    responder Seller
    artifact RequestForInformation
    artifact Catalog
    artifact PurchaseOrder
    artifact PurchaseOrderConfirmation
    artifact Payment
}
    
```

Figure 6: Sample Interaction Model in InteractionDSL

Regarding tools for model transformation, widely used ones are eclipse/ATL and QVT. They could also be used to execute model transformations described in this paper.

The Xpand template used is shown in Fig. 7, which should be considered as a sample. It does the following: a) import metamodel, b) import helper functions, c) declare this as a transformation against Process, d) specify output file, e) define transformation processing for each lane, f) take only steps passing artifact to other lanes, g) generate Service Interfaces, h) generate Participants, and i) generate Service Contracts with model information.

```

@Model:
    processes+=Process*;
@Process:
    'process' name=ID '{'
    lanes+=Lane*
    '}' ;
@Lane:
    'role' name=ID '{'
    processFlows+=ProcessFlow*
    '}' ;
@ProcessFlow:
    InitialNode | EndNode | Step | Artifact | ParallelSplit |
    ParallelMerge | ConditionalSplit | ConditionalMerge ;
InitialNode: 'start' name=FQN ;
EndNode: 'end' name=FQN;
@Step: 'step' name=FQN ('{'
    ('description' description=STRING)?
    fromNode=FromNode
    toNode=ToNode
    '}')? ;
@Artifact: 'artifact' name=FQN ('{'
    ('description' description=STRING)?
    fromNode=FromNode
    toNode=ToNode
    '}')? ;
@ParallelSplit: 'parallel' 'split' name=FQN ('{'
@ParallelMerge: 'parallel' 'merge' name=FQN ('{'
@ConditionalSplit: 'conditional' 'split' name=FQN ('{'
@ConditionalMerge: 'conditional' 'merge' name=FQN ('{'
@FromNode:
@ToNode:
FQN: ID('.'ID)* ;
    
```

Figure 4: Partial ProceDSL grammar

```

process Buy_Sell_Ship {
    role buyer {
        start StartOfPurchasingProcess
        step RequestForInformation {
            description "Gathering Product Information"
            from StartOfPurchasingProcess
            to seller.ReceiveRFI with Artifacts.RFI
        }
        step ReceiveCatalog {}
        conditional split PurchaseFromCatalog {}
        step SendQuoteRequest {}
        step ReceiveQuote {}
        step SendPO {}
        step ReceivePOConfirmation {}
        parallel merge MergeWithInformation {}
        step ReceiveShipmentInformation {}
        parallel merge MergeWithProductsDelivery {}
        step ReceiveProducts {}
        step PaymentToSeller {}
        end EndOfPurchasingProcess
    }
    role seller {}
    role shipper {}
    role Artifacts {
        artifact RFI {
            description "Request for information document"
            from buyer.RequestForInformation
            to seller.ReceiveRFI
        }
        artifact Catalog {}
        artifact QuoteRequest {}
        artifact Quote {}
        artifact PO {}
        artifact POConfirmation {}
        artifact ShippingRequest {}
        artifact ShippingCostEstimate {}
        artifact ShipmentInformation {}
        artifact Products {}
        artifact SellerPayment {}
        artifact ShipperPayment {}
    }
}
    
```

Figure 5: Sample Process Model in ProcessDSL

```

«IMPORT jp::ac::fun::xtext::process::processDsl»
«EXTENSION templates::Extensions»
«REM» SoaML Service Architecture «ENDREM»
«DEFINE main FOR Process-»
«FILE this.name+ ".soa"-»
ServiceArchitecture «this.name» {
    «FOREACH this.lanes AS lane-»
        «FOREACH lane.processFlows.typeSelect(Step) AS node-»
            «IF node.fromNode.fnode.eContainer() == node.eContainer() -»
                «ELSE-»
                    «IF node.fromNode.isArtifactProvidedFrom() -»
                        ServiceInterface «node.fromNode.fnode.name» {
                            }
                        «ENDIF-»
                    «ENDIF-»
                «ENDFOREACH-»
            «ENDFOREACH-»
        «FOREACH this.lanes AS lane» «IF lane.name == "Artifacts"»
            «ELSE-»
                Participant «lane.name» {
                    }
                «ENDIF-»
            «ENDFOREACH-»
        «FOREACH this.lanes AS lane»
            «IF lane.name == "Artifacts" -»
                «FOREACH lane.processFlows AS msg-»
                    Message «msg.name»
                «ENDFOREACH-»
            «ENDIF-»
        «ENDFOREACH-»
        «FOREACH this.lanes AS lane»
            «FOREACH lane.processFlows.typeSelect(Step) AS node-»
                «IF node.toNode.tnode.eContainer() == node.eContainer() -»
                    «ELSE-»
                        «IF node.toNode.isArtifactProvided() -»
                            ServiceContract «node.name» "" among {
                                consumer «node.eContainer().getContainerName()»
                                    AsConsumer
                                    performed by «node.eContainer().getContainerName()»
                                provider «node.toNode.tnode.eContainer().getContainerName()»
                                    AsProvider
                                    performed by «node.toNode.tnode.eContainer().getContainerName()»
                                «node.eContainer().getContainerName()» AsConsumer
                            -> «node.toNode.tnode.eContainer().getContainerName()» AsProvider
                                «node.toNode.tnode.name»
                            }
                        «ENDIF-»
                    «ENDIF-»
                «ENDFOREACH-»
            «ENDFOREACH-»
        «ENDIF-»
    «ENDIF-»
«ENDFILE-»
«ENDEFFINE»
    
```

Figure 7: Sample model transformation definitions

As shown, if the source model contains enough information with formally defined grammar or metamodel, it is possible to create text file based on available information and by navigating the model elements. We have generated a small number of text files using this method, and will examine those in the next section.

One question may be asked about how to make sure that the DSL targeting only core concepts can be used in the research like this. It is different, but if it contains major elements with right relationships, it is possible to compare the grammar or generated EMF ecore file with that of full modeling language to see if there is any major inconsistencies or issues in using it in the research.

9 RESULTS OF PROCESS TO SERVICE TRANSFORMATION

Now let us examine what the transformation produced. The first is a result around Service Architecture.

A. Participant/Service Contact/Service Architecture

From the used template above, here is a summary of what operations were given to the sample process model.

- 1) Service Architecture name is derived from Process's name.
- 2) Participant name is derived from lane's name.
- 3) Service Contract is defined for the step that passes Artefact across the lanes. Other types of actions will be discarded. Service Contract name is derived from the step name that initiates interaction.

If we modify the grammar to include marking to show the logical boundary of the application, it may become possible to have coarse grained Service Contract with multiple Artefact, but that will introduce another requirement on the dependencies between Artefact.

The generated textual model is shown in Fig. 8. It is generated from sample process model in ProcessDSL (Figure 5) by applying the process to service model transformation (Figure 7). Note that the model data below has been imported into ServiceDSL editor.

B. Service Interface

Service Interface in SoaML is functional elements, and is more like interface and signatures in Computational Object. The best way is to define Computational Object with process definition, but that will lead to a different world. Based on the generated Service Contract, it is safe to assume that Participant on receiving side have capability to process the Artefact passed from the other side. This implies that there exists Service Interface on the Service Provider Component. However there is no information about signatures in process models in general, it would not be possible to generate Message elements either. Therefore the rule applied was Service Interface is derived from the node (name) that receives Artifact.

C. Service Choreography

Service Choreography is a set of defined sequence of service interactions between the two Participants, which is specified using UML Sequence Diagram. This contrasts

with UML Activity Diagram we used to specify process. Although not included in Figure 8, it is possible to collect interactions between different lanes in the process diagram.

Although BPMN, Activity Diagram, and SoaML are all graphical modeling language, we applied our method to define simplified textual DSL, and were able to transform a process model to a service model, although in a limited manner.

```

ServiceArchitecture Buy_Sell_Ship {
+ ServiceInterface SendCatalog {}
+ ServiceInterface SendQuoteResponse {}
+ ServiceInterface AcceptPO {}
+ ServiceInterface RequestForInformation {}
+ ServiceInterface SendQuoteRequest {}
+ ServiceInterface SendPO {}
+ ServiceInterface SendEstimate {}
+ ServiceInterface PaymentToSeller {}
+ ServiceInterface ReqeustShipping {}
+ ServiceInterface PaymentToShipper {}
+ Participant buyer {}
+ Participant seller {}
+ Participant shipper {}
Message RFI
Message Catalog
Message QuoteRequest
Message Quote
Message PO
Message POConfirmation
Message ShippingRequest
Message ShippingCostEstimate
Message ShipmentInformation
Message Products
Message SellerPayment
Message ShipperPayment
+ ServiceContract RequestForInformation "" among {}
+ ServiceContract SendQuoteRequest "" among {}
+ ServiceContract SendPO "" among {}
+ ServiceContract PaymentToSeller "" among {}
+ ServiceContract SendCatalog "" among {}
+ ServiceContract SendQuoteResponse "" among {}
+ ServiceContract AcceptPO "" among {}
+ ServiceContract ReqeustShipping "" among {}
+ ServiceContract ReportToBuyer "" among {}
+ ServiceContract PaymentToShipper "" among {}
+ ServiceContract SendEstimate "" among {}
+ ServiceContract ShipProducts "" among {}
}

```

Figure 8: Sample Generated Service Architecture

10 SERVICE TO PROCESS TRANSFORMATION

In this section we will start with service model definition. The first step is to define the grammar for ServiceDSL that implements SoaML's core concepts, which are Service Architecture, Service Contract, Participant and Service Interface. Again, here is a portion of the grammar definition (more than ten elements are not shown in Fig. 9).

This DSL is simple enough to cover the structural aspects of SoaML model, and we even tried to include behavioral (sequence diagram) aspect in the grammar.

Next thing is to create a service model based on this DSL, which is shown in Fig. 10, which is done also by typing

The previous Fig. 2 showed a graphical representation of a sample Service Architecture at very high level. In the textual ServiceDSL model of Fig. 10, we included major elements under Service Architecture, because Service Architecture works as a root of the model in this language. Each usage of the typical language element is shown at least once, but not all the elements are shown by using the folding option to make the Figures smaller to fit in this paper. Also

note that message sequencing is specified in Service Contract, e.g. with “buyer -> seller RFI optional,” implementing message flows, or cross-lane object nodes, described in Fig. 1.

```

@Model:
  (elements+=Type)*;
@Type:
  DataType | SOACollaboration ;
@DataType:
  'type' name=ID;
@FQN:
  ID ('.' ID)*;
@SOACollaboration:
  'ServiceArchitecture' name=ID '{'
    (soaInterfaces+=SOAInterface)*
    (soaParticipants+=SOAParticipant)*
    (soaChannels+=SOAChannel)*
    (soaMessages+=SOAMessage)*
    (soaContracts+=ServiceContract)*
  '}' ;
@SOAInterface:
  'ServiceInterface' name=ID '{'
    interfaces+=Interface*
  '}' ;
Interface: ProvidedInterface | RequiredInterface ;
@ProvidedInterface:
  'ProvidedInterface' name=ID '{'
    (params+=Param)*
  '}' ? ;
@RequiredInterface:[]
@Param:
  'Parameter' dir=Direction name=ID ':' type=[DataType] ;
@SOAParticipant :
  'Participant' name=ID '{'
    (interactionPoints+=InteractionPoint)*
  '}' ;
@InteractionPoint:
  ServicePoint | RequestPoint ;
@ServicePoint:
  'ServicePoint' name=ID ':' ref=[SOAInterface|ID] ;
@RequestPoint:[]
@ServiceContract:
  'ServiceContract' name=ID description=STRING 'among' '{'
    roles+=Role*
    sequences+=Sequence*
  '}' ;
    
```

Figure 9: Partial ServiceDSL grammar

The major difference between this service model description and the previous process model description is in the style of control flow description, i.e. sequence versus activity, and the number of parties involved, i.e. only two parties in service model vs. possibly more than three parties in process model. In service model, service is the central concept and therefore major players in service model are consumer and provider. On the other hand, in process model, the focus is on control flow, object flow, conditional or parallel split and merge, covering all the players that could be more than three players. It is clear that service model is not able to express e.g. control flow within the same lane in process model, since they are not exposed as service interaction and of no interest in service model.

We can still apply model transformation to see what we can get even though the limitation is clear. Figure 11 shows a sample Xpand template to transform service model to process model. Figure 12 shows a transformed sample process based on the service model (Figure 10). It seems step portions of the process were successfully generated. But these are just concatenation of the sequences from Service Contract’s sequence definitions.

If full control flow needs to be generated from the service description, process oriented description should be a part of the service model. In SoaML specification, these process aspects are treated as requirements specification to services, and therefore they are outside the scope of SoaML language itself (no stereotype or reference is defined as mandatory

against Activity). The authors are planning to submit comments to OMG (or ISO if it is proposed) to clarify and enhance the standard or specification.

```

ServiceArchitecture BuySellShip {
  ServiceInterface BuyInterface {
    RequiredInterface RFI {
      Parameter in RequiredProducts : String
      Parameter out Information : Document
    }
    RequiredInterface PO {}
    RequiredInterface PaySeller {}
  }
  ServiceInterface ReceiveProduct {}
  ServiceInterface SellInterface {}
  ServiceInterface ShipperInterface {}
  Participant Buyer {
    RequestPoint RFI : BuyInterface
    RequestPoint PO : BuyInterface
    RequestPoint Payment : BuyInterface
    ServicePoint Products : ReceiveProduct
  }
  Participant Seller {}
  Participant Shipper {}
  ServiceChannel BuySell {}
  ServiceChannel SellShip {}
  ServiceChannel ShipBuy {}
  Message RFI
  Message Catalog
  Message PO
  Message POConfirmation
  Message RequestShipping
  Message EstimateForShipping
  Message ShippingStatus
  Message DeliverProduct
  Message SellerPayment
  Message ShipperPayment
  ServiceContract PlaceOrder "Ordering Products" among {
    consumer buyer performed by Buyer
    provider seller performed by Seller
    buyer->seller RFI optional
    buyer<-seller Catalog
    buyer->seller PO
    buyer<-seller POConfirmation
    buyer->seller SellerPayment
  }
  ServiceContract RequestShipping "Seller request for shipment" among {}
  ServiceContract DeliverProducts "Shipper delivers products" among {}
}
    
```

Figure 10: Sample Service Architecture

```

«IMPORT jp::ac::fun::xtext::soa::soaDel»
«EXTENSION templates::Extensions»
«REM» Generating Process Steps «ENDREM»
«DEFINE main FOR SOACollaboration-»
«FILE name+ ".process"-»
process «this.name» {
  «FOREACH this.soaParticipants AS participant-»
    role «participant.name» {
      «FOREACH this.soaContracts AS contract-»
        «FOREACH contract.sequences AS seq-»
          «IF seq.consumer.performer==participant-»
            «IF seq.arrow=="->"-»
              step Request«seq.message.name»
            «ELSE-»
              step Receive«seq.message.name»
            «ENDIF-»
          «ELSEIF seq.provider.performer==participant-»
            «IF seq.arrow=="->"-»
              step Receive«seq.message.name»
            «ELSE-»
              step Request«seq.message.name»
            «ENDIF-»
          «ENDIF-»«ENDFOREACH-»«ENDFOREACH-»
        }
      «ENDIF-»
    }
  role Artifacts {
    «FOREACH this.soaMessages AS msg-»
      artifact «msg.name»
    «ENDIF-»
  }
}
«ENDFILE»«ENDDDEFINE»
    
```

Figure 11: Sample model transformation definitions

```

process BuySellShip {
  role Buyer {
    step RequestRFI
    step ReceiveCatalog
    step RequestPO
    step ReceivePOConfirmation
    step RequestSellerPayment
    step RequestShippingStatus
    step ReceiveDeliverProduct
  }
  role Seller {
    step ReceiveRFI
    step RequestCatalog
    step ReceivePO
    step RequestPOConfirmation
    step ReceiveSellerPayment
    step RequestRequestShipping
    step ReceiveEstimateForShipping
    step RequestShipperPayment
  }
  role Shipper {
    step ReceiveRequestShipping
    step RequestEstimateForShipping
    step ReceiveShipperPayment
    step ReceiveShippingStatus
    step RequestDeliverProduct
  }
  role Artifacts {
    artifact RFI
    artifact Catalog
    artifact PO
    artifact POConfirmation
    artifact RequestShipping
    artifact EstimateForShipping
    artifact ShippingStatus
    artifact DeliverProduct
    artifact SellerPayment
    artifact ShipperPayment
  }
}

```

Figure 12: Sample Transformed Process

11 EVALUATION

For this specific buy-sell-ship example, we counted lines of text for each models. Manually written ProcessDSL-based model contained 227 lines, and ServiceDSL-based model transformed from it contained 88 lines. Manually written ServiceDSL-based model contained 125 lines, and ProcessDSL-based model transformed from it contained 40 lines. If both manually written models are semantically equivalent or very close, each transformed model should be reasonably compatible with the other manually written model. The line numbers comparison rates are 70.4% for Process to Service transformation, and 17.6% for Service to Process transformation. Although line by line comparison is better, this gives an implication that Process to Service transformation works much better than the other way. For instance, a portion of manually written ServiceDSL-based model below (Figure 13) is compared by a portion of generated model from ProcessDSL-based model (Figure 14).

```

ServiceContract PlaceOrder "Ordering Products" among {
  consumer buyer performed by Buyer
  provider seller performed by Seller
  buyer->seller RFI optional
  buyer<-seller Catalog
  buyer->seller PO
  buyer<-seller POConfirmation
  buyer->seller SellerPayment
}

```

Figure 13: Manually written Service Contract

```

ServiceContract RequestForInformation "" among {
  consumer buyerAsConsumer performed by buyer
  provider sellerAsProvider performed by seller
  buyerAsConsumer->sellerAsProvider RFI
}
ServiceContract SendQuoteRequest "" among {
  consumer buyerAsConsumer performed by buyer
  provider sellerAsProvider performed by seller
  buyerAsConsumer->sellerAsProvider QuoteRequest
}
ServiceContract SendPO "" among {
  consumer buyerAsConsumer performed by buyer
  provider sellerAsProvider performed by seller
  buyerAsConsumer->sellerAsProvider PO
}
ServiceContract PaymentToSeller "" among {
  consumer buyerAsConsumer performed by buyer
  provider sellerAsProvider performed by seller
  buyerAsConsumer->sellerAsProvider SellerPayment
}
ServiceContract SendCatalog "" among {
  consumer sellerAsConsumer performed by seller
  provider buyerAsProvider performed by buyer
  sellerAsConsumer->buyerAsProvider Catalog
}
ServiceContract SendQuoteResponse "" among {
  consumer sellerAsConsumer performed by seller
}

```

Figure 14: Transformed/Generated Service Contract

This shows that if there is good semantic relationship, we can get reasonable transformation result. The result will depend on how much of such good semantic relations with direction, e.g. Process to Service, exist between the two architecture styles.

12 RELATION WITH DISTRIBUTED COMPUTING

If we look at both process oriented and service oriented models in the context of distributed computing, they can be considered as candidate sources of execution on the platform, which may be internal enterprise systems or hybrid with cloud computing platform environment or full public cloud platform.

There is a category of technology called process engines that interpret and execute process definitions. Workflow engines are also considered as ones in this category. Their focus, however, is on controlling and monitoring the given process flow, and not on the execution of distributed services. Regarding SOA, Web Services is one of the typical implementation technologies, and they can be considered as base technology for distributed computing. With the use of SoaML, most of the necessary information to map down to SOA implementations is included in the model, and therefore it is not surprising to find products to do code generation based on SoaML model and actually run on the SOA runtime platform.

Our interest here is how close to implementation we can get based on process model via service model. We created a transformation template (not included in this paper) to generate skeleton interface codes of the service components. The result is shown in Fig. 15. As expected, there is not much detailed information included, since some of the control information is discarded when converting it to service model. To make this code work, it needs to be completed with more detailed information with implementation classes, WSDL files, and frameworks for SOA such as eclipse SOA Platform etc.


```

package jp.ac.fun.xtext.services;

import javax.jws.*;

public interface SendCatalogService {
    public void SendCatalog (
    );
}

public interface AcceptPOService {
    public void AcceptPO (
    );
}

public interface RequestForInformationService {
    public void RequestForInformation (
    );
}

public interface SendPOService {
    public void SendPO (
    );
}

public interface SendEstimateService {
    public void SendEstimate (
    );
}

public interface PaymentToSellerService {
    public void PaymentToSeller (
    );
}

public interface RequestShippingService {
    public void RequestShipping (
    );
}

public interface PaymentToShipperService {
    public void PaymentToShipper (
    );
}
    
```

Figure 15: Generated Skeleton Code

13 CONCLUSIONS

DSLs are usually used at area close to programming. We demonstrated that DSLs, which captures only core concepts of the target modeling language, can be applied to architecture descriptions that are Process-oriented and Service-oriented architectures and can be used to examine the difference between the two styles of modeling presented in this paper.

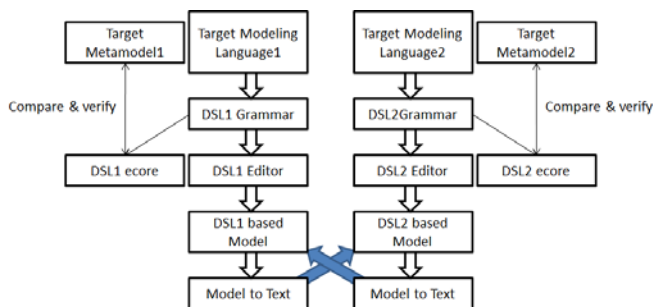


Figure 16: Model Transformation using M2T transformation

We created textual DSLs, including ProcessDSL and ServiceDSL, and showed sample transformations from process model to service model, and service model to process model. In doing so, we found a major difference between process modeling and service modeling. Something internal in process modeling will be lost when it is converted into service model, e.g. internal control flow. Orchestration of all the participants, which is the essential part of process model, is not possible in most cases when transforming service model to process model, since services are only meaningful to consumers and providers, i.e. between two parties, and normally orchestration aspects are left to higher level activities.

There is also a fundamental difference between the two, which is about level of abstraction. In process modeling, the level of abstraction is at end users or at business analysts level, but service modeling further includes interface specifications that are at architects’ or developers’ viewpoint. This caused transformation loss from service to process, and also was the reason of insufficient output from process model.

The possibility of service interface generation from process model was examined and only skeleton interface codes were generated because of the semantic gap between the two models with associated information loss in transformation. This does not, however, preclude the possibility of code generations from process model into process engines’ environment and from service model into SOA environment.

Based on above, authors believe that they are showing the different aspects of the business system, and if mixed use is required, positioning process model as higher than service model will better work in enterprise architectures than positioning them in the opposite order or placing them at the same level.

Now, let us consider cost/performance of this project. The use case is an enterprise project to integrate its BPM based system with its SOA based system from multiple vendors or from multiple departments or by the result of M&A. If we apply the transformation template to this BPM based system, which is just automated transformation, we can get a list of candidate services required to implement the functionality of the BPM-based system. If this list is compared with existing SOA based system’s services list, it is very likely to find similarities and missing pieces necessary for new integrated system. The merit would become clearer with the system size grows. Even though initial investment is required, this type of research will bring actual benefits in this kind of use cases.

Regarding the tooling, eclipse/Xtext provided necessary DSL development environment, and integrated model to text transformation facility Xpand/Xtend worked well to generate text from the DSL based models.

14 FUTURE WORKS

There are some areas where we need further works.

We will need to investigate a mechanism to verify the created DSLs, a mechanism to store trace/log information in transformation, more resources like practical examples around DSLs and model transformations, and more specialized tools to achieve specific activities.

We will also need to experiment on minimizing transformation loss such as internal steps in a process, possibly by introducing control flow manager in each lane so that each step could be transformed to internal service etc. or by introducing service orchestration function to the service model.

We are planning to use full RM-ODP model as source in the next experiment to see how component definitions in computational specification could contribute to process to service model transformations.

ACKNOWLEDGEMENTS

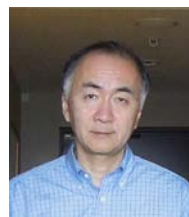
The authors thank contributors to eclipse modeling projects, especially to eclipse/Xtext open source project, and further thanks OMG and contributors of many modeling standards including UML, BPMN and SoaML. The authors also thank many people from various countries involved in ISO/IEC JTC1/SC21/WG7, SC7/WG19, and ITU-T SG19's continuing efforts to standardize and maintain RM-ODP family of standards.

REFERENCES

- [1] J. A. Zachman, "A Framework for Information Systems Architecture," IBM Systems Journal, Vol.26, No.3, p.276 (1987).
- [2] FEA Consolidated Reference Model Document Version 2.3, http://www.whitehouse.gov/sites/default/files/omb/assets/fea_docs/FEA_CRM_v23_Final_Oct_2007_Revised.pdf (2007).
- [3] OMG, "Business Process Model and Notation (BPMN) Version 2.0," <http://www.omg.org/spec/BPMN/2.0/PDF/> (2011).
- [4] OMG, "Semantics Of Business Vocabulary And Business Rules (SBVR), Version 1.0," <http://www.omg.org/spec/SBVR/1.0/PDF/> (2008).
- [5] D. Luckham, "The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems," Addison-Wesley (2002).
- [6] OMG, "Unified Modeling Language (UML)," <http://www.omg.org/spec/UML/> (2011).
- [7] ISO/IEC 10746-2:2009, Information technology - Open distributed processing -- Reference model: Foundations
ISO/IEC 10746-3:2009, Information technology - Open distributed processing -- Reference model: Architecture
ISO/IEC 15414:2006, Information technology -- Open distributed processing -- Reference model -- Enterprise language
ISO/IEC 19793:2008, Information technology -- Open Distributed Processing -- Use of UML for ODP system specifications.
- [8] M. Fowler, "Domain-Specific Languages," Addison-Wesley (2011).
- [9] A. Kleppe, "Software Language Engineering," Addison-Wesley (2009).
- [10] OASIS, "Reference Model for Service Oriented Architecture 1.0," <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf> (2006).
- [11] OMG, "Service oriented architecture Modeling Language (SoaML) Specification," <http://www.omg.org/spec/SoaML/1.0/> (2012).
- [12] OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification," <http://www.omg.org/spec/QVT/1.1/PDF/> (2011).
- [13] F. Jouault, F. Allilaire, J. Bezivin and I. Kurtev, "ATL: A model transformation tool," Science of Computer Programming, Vol.72, No.1-2, Special Issue on Second issue of experimental software and toolkits (EST), pp.31-39 (2008).
- [14] Eclipse, "ATL Transformations," <http://projects.eclipse.org/projects/modeling.mmt.atl>.
- [15] OMG, "MOF 2 XMI Mapping Specification," <http://www.omg.org/spec/XMI/> (2011).
- [16] Eclipse Foundation, Xtext - Language Development Made Easy!, <http://www.eclipse.org/Xtext/>.
- [17] The Spoofox Language Workbench, <http://strategoxt.org/Spoofox>.
- [18] JetBrains : Meta Programming System - Language Oriented Programming environment and DSL creation tool, <http://www.jetbrains.com/mps/>.
- [19] Xtext Users Japan, <http://sites.google.com/site/xttextusersjapan/files>.

(Received February 27, 2012)

(Revised February 28, 2013)



Akira Tanaka received M.E. from University of Tokyo, Japan in 1978. From 1978 to 2008, he was with Hitachi Ltd. His current research area includes model-based software development. He is a member of ACM, IEEE Computer Society and IPSJ.



Osamu Takahashi received master's degree from Hokkaido University in 1975. He is currently a professor at the Department of System Information Science at Future University Hakodate, Japan. His research interest includes ad-hoc net-work, network security, and mobile computing. He is a member

of IEEE, IEICE, and IPSJ.