Deriving Semantics from WS-BPEL Specifications of Parallel Business Processes on an Example

Prof., Dr. Vladimir Dimitrov, University of Sofia, Faculty of Mathematics and Informatics

**Abstract**: For today distributed and parallel systems, a widely accepted standard for specification of business processes is WS-BPEL. This standard is mismatch of algebraic and Petri net models, and as result of that it is easy possible to write business process with deadlocks and other not wanted features. That is why verification of business processes is very important topic in software engineering. The intent of this paper is to show possibilities for conversion of a WS-BPEL process into more formal models that can be formally verified. As formal models are used CSP and Z-notation. The last one is useful for specification of abstract data types. Web services can be viewed as a kind of last ones.

# Motivation

There are two kinds of business processes in WS-BPEL [1]: executable and abstract ones.

Behavioral semantics of executable business processes is well defined in the notation framework. There is a problem with extensions, because they go outside the notation framework and they are open and unpredictable.

The other category are abstract business processes. Their intention is to describe Web services interactions without internal implementation details. The standard defines concretization procedure that creates an executable business process from an abstract one. The problem here is that it is possible to be generated an executable business process with behavior different from that of the abstract one. It is possible, the executable business process to contain interactions changes of the abstract one and that are not simply its specialization. The standard requires for every abstract process to exist at least one executable business process that is concretized by the procedure defined in the standard and that is compatible with the abstract one. In such a way, the standard guarantees that above mentioned deviations are not available for at least one executable business process.

Abstract business process represents a class of executable business processes compatible with it. It is more productive to investigate abstract business processes because:

1. Investigating abstract business process means investigation of the whole class of executable business processes that it represents.
2. Abstract business process does not contain implementation details that have no impact on Web services interactions.

WS-BPEL business process has two parts. First one is the WSDL [2] specification of the Web services involved in the interaction. Usually, this specification includes specification of the business process as a Web service. The second part is the WS-BPEL specification of the business process as Web services interactions. These both specifications are complementary because the business process often is a Web service that is specified in WSDL. On the other hand, the business process as Web service is implemented in WS-BPEL. Web services extensions to WSDL specification for Web Services participating in the business process are essential ones for the WS-BPEL specification.

WSDL specifications of Web services are specifications of interfaces that hide implementation details. These specification could be formalized. Why such a formalization is needed? WSDL is XML based notation. Authors of XML argue that this notation is readable by humans and computers. The truth is that XML specifications are verbose and they not readable by the target readers. That is why it is preferable if Web services are formalized in some established notation that is well recognized by humans. Such a tool is the Z notation [3]. Specifications in the last one tends to be very compact.

Z notation is mainly used for specification of abstract data types. It can be used specification of algebraic notations too.

Web services represented in WSDL could be viewed as abstract data types. In the example below the WSDL and WS-BPEL specifications are a specification of abstract business process taken from the standard.

# Formalization of the WSDL specification

First, messages exchanged among Web services are defined (shippingPT.wsdl):

<wsdl:definitions

 targetNamespace="http://example.com/shipping/interfaces/"

 xmlns:ship="http://example.com/shipping/ship.xsd"

 xmlns:tns="http://example.com/shipping/interfaces/"

 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <wsdl:types>

 <xsd:schema>

 <!-- import ship schema -->

 </xsd:schema>

 </wsdl:types>

 <wsdl:message name="shippingRequestMsg">

 <wsdl:part name="shipOrder" type="ship:shipOrder" />

 </wsdl:message>

 <wsdl:message name="shippingNoticeMsg">

 <wsdl:part name="shipNotice" type="ship:shipNotice" />

 </wsdl:message>

 <wsdl:portType name="shippingServicePT">

 <wsdl:operation name="shippingRequest">

 <wsdl:input message="tns:shippingRequestMsg" />

 </wsdl:operation>

 </wsdl:portType>

 <wsdl:portType name="shippingServiceCustomerPT">

 <wsdl:operation name="shippingNotice">

 <wsdl:input message="tns:shippingNoticeMsg" />

 </wsdl:operation>

 </wsdl:portType>

</wsdl:definitions>

In this part of the WSDL specification are imported application data schemas. From the last ones types shipOrder and shipNotice are used. They are application data containers. Only the properties defined on these messages have impact on the message exchange among the Web services. These data types are modeled as basic types in Z notation:



Messages are modelled with Z schemas:





Every message part is defined with a field with the same name and same type in the corresponding message Z schema.

Port types define Web services. They could be represented as abstract data types. In Z notation, abstract data types are defined with schema type (Z schema) and operations (Z schemas) applied on it. There is no way in Z notation, operations to be defined on basic types. That is why initially, the basic type WebService is introduced and then it is used in the port types Z schemas. The field ws not very elegant approach for introducing Web services, but, never mind, it works.

All port types are only with one operation that is represented with the corresponding Z schema. Operations are one way. Each of them has only one input parameter.



These specifications of the Web services do not contain any information about the Web services structure or behavior.

Properties definition in the WSDL specification is:

<wsdl:definitions

 targetNamespace="http://example.com/shipping/properties/"

 xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"

 xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"

 xmlns:ship="http://example.com/shipping/ship.xsd"

 xmlns:sif="http://example.com/shipping/interfaces/"

 xmlns:tns="http://example.com/shipping/properties/"

 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <wsdl:import location="shippingPT.wsdl"

 namespace="http://example.com/shipping/interfaces/" />

 <!-- types used in Abstract Processes are required to be finite

 domains. The itemCountType is restricted by range -->

 <wsdl:types>

 <xsd:schema

 targetNamespace="http://example.com/shipping/ship.xsd">

 <xsd:simpleType name="itemCountType">

 <xsd:restriction base="xsd:int">

 <xsd:minInclusive value="1" />

 <xsd:maxInclusive value="50" />

 </xsd:restriction>

 </xsd:simpleType>

 </xsd:schema>

 </wsdl:types>

 <vprop:property name="shipOrderID" type="xsd:int" />

 <vprop:property name="shipComplete" type="xsd:boolean" />

 <vprop:property name="itemsTotal" type="ship:itemCountType" />

 <vprop:property name="itemsCount" type="ship:itemCountType" />

 <vprop:propertyAlias propertyName="tns:shipOrderID"

 messageType="sif:shippingRequestMsg" part="shipOrder">

 <vprop:query>

 ship:ShipOrderRequestHeader/ship:shipOrderID

 </vprop:query>

 </vprop:propertyAlias>

 <vprop:propertyAlias propertyName="tns:shipOrderID"

 messageType="sif:shippingNoticeMsg" part="shipNotice">

 <vprop:query>ship:ShipNoticeHeader/ship:shipOrderID</vprop:query>

 </vprop:propertyAlias>

 <vprop:propertyAlias propertyName="tns:shipComplete"

 messageType="sif:shippingRequestMsg" part="shipOrder">

 <vprop:query>

 ship:ShipOrderRequestHeader/ship:shipComplete

 </vprop:query>

 </vprop:propertyAlias>

 <vprop:propertyAlias propertyName="tns:itemsTotal"

 messageType="sif:shippingRequestMsg" part="shipOrder">

 <vprop:query>

 ship:ShipOrderRequestHeader/ship:itemsTotal

 </vprop:query>

 </vprop:propertyAlias>

 <vprop:propertyAlias propertyName="tns:itemsCount"

 messageType="sif:shippingRequestMsg" part="shipOrder">

 <vprop:query>

 ship:ShipOrderRequestHeader/ship:itemsCount

 </vprop:query>

 </vprop:propertyAlias>

 <vprop:propertyAlias propertyName="tns:itemsCount"

 messageType="sif:shippingNoticeMsg" part="shipNotice">

 <vprop:query>ship:ShipNoticeHeader/ship:itemsCount</vprop:query>

 </vprop:propertyAlias>

</wsdl:definitions>

A new type for the properties is introduced and it Z schema is:



The properties are then represented as types:

There are two deviations from the WSDL specification. The order numbers are positive numbers – not simply integers as is defined in the WSDL specification. In Z notation, there is no Boolean type and it is modelled with two values False and True.

Aliases are properties placed in messages. Here, they are modelled as functions from message type to property type. There is no need to model XPath queries, because the last ones are standard extensions to WS-BPEL. Modelling aliases as functions permits queries written in other languages to be modelled in the same way. The specification of aliases is more abstract:



One property could have many aliases with the same name. In Z notation, above defined functions are global ones and their names must be unique. So, alias name is formed by the property name and part name, in which it is defined. It is mapping from message type to property type.

Formalization of partner link type has no sensible interpretation here and they are modeled in the context of the business process.

Finally, as result of the modelling effort, the specification is very compact and very simple. It does not contain the business process as a Web service. This specification could be used for software development, but it is very simple.

# Formalization of the WS-BPEL specification

The specification of the example abstract business process in WS-BPEL is:

<process name="shippingService"

 targetNamespace="http://example.com/shipping/"

 xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/abstract"

 xmlns:plt="http://example.com/shipping/partnerLinkTypes/"

 xmlns:props="http://example.com/shipping/properties/"

 xmlns:ship="http://example.com/shipping/ship.xsd"

 xmlns:sif="http://example.com/shipping/interfaces/"

 abstractProcessProfile=

 "http://docs.oasis-open.org/wsbpel/2.0/process/abstract/ap11/2006/08">

 <import importType="http://schemas.xmlsoap.org/wsdl/"

 location="shippingLT.wsdl"

 namespace="http://example.com/shipping/partnerLinkTypes/" />

 <import importType="http://schemas.xmlsoap.org/wsdl/"

 location="shippingPT.wsdl"

 namespace="http://example.com/shipping/interfaces/" />

 <import importType="http://schemas.xmlsoap.org/wsdl/"

 location="shippingProperties.wsdl"

 namespace="http://example.com/shipping/properties/" />

 <partnerLinks>

 <partnerLink name="customer" partnerLinkType="plt:shippingLT"

 partnerRole="shippingServiceCustomer"

 myRole="shippingService" />

 </partnerLinks>

 <variables>

 <variable name="shipRequest" messageType="sif:shippingRequestMsg" />

 <variable name="shipNotice" messageType="sif:shippingNoticeMsg" />

 <variable name="itemsShipped" type="ship:itemCountType" />

 </variables>

 <correlationSets>

 <correlationSet name="shipOrder" properties="props:shipOrderID" />

 </correlationSets>

 <sequence>

 <receive partnerLink="customer" operation="shippingRequest" variable="shipRequest">

 <correlations>

 <correlation set="shipOrder" initiate="yes" />

 </correlations>

 </receive>

 <if>

 <condition>

 bpel:getVariableProperty('shipRequest', 'props:shipComplete')

 </condition>

 <sequence>

 <assign>

 <copy>

 <from variable="shipRequest" property="props:shipOrderID" />

 <to variable="shipNotice" property="props:shipOrderID" />

 </copy>

 <copy>

 <from variable="shipRequest" property="props:itemsCount" />

 <to variable="shipNotice" property="props:itemsCount" />

 </copy>

 </assign>

 <invoke partnerLink="customer" operation="shippingNotice" inputVariable="shipNotice">

 <correlations>

 <correlation set="shipOrder" pattern="request" />

 </correlations>

 </invoke>

 </sequence>

 <else>

 <sequence>

 <assign>

 <copy>

 <from>0</from>

 <to>$itemsShipped</to>

 </copy>

 </assign>

 <while>

 <condition>

 $itemsShipped &lt; bpel:getVariableProperty('shipRequest', 'props:itemsTotal')

 </condition>

 <sequence>

 <assign>

 <copy>

 <opaqueFrom/>

 <to variable="shipNotice" property="props:shipOrderID" />

 </copy>

 <copy>

 <opaqueFrom/>

 <to variable="shipNotice" property="props:itemsCount" />

 </copy>

 </assign>

 <invoke partnerLink="customer" operation="shippingNotice" inputVariable="shipNotice">

 <correlations>

 <correlation set="shipOrder" pattern="request" />

 </correlations>

 </invoke>

 <assign>

 <copy>

 <from>

 $itemsShipped + bpel:getVariableProperty('shipNotice', 'props:itemsCount')

 </from>

 <to>$itemsShipped</to>

 </copy>

 </assign>

 </sequence>

 </while>

 </sequence>

 </else>

 </if>

 </sequence>

</process>

At the beginning partner link, in which the business process participates, is defined. The role of the process in this link is fixed. The partner link is defined with the partner link type from the WSDL specification (shippingLT.wsdl):

<wsdl:definitions

 targetNamespace="http://example.com/shipping/partnerLinkTypes/"

 xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"

 xmlns:sif="http://example.com/shipping/interfaces/"

 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

 <wsdl:import location="shippingPT.wsdl"

 namespace="http://example.com/shipping/interfaces/" />

 <plnk:partnerLinkType name="shippingLT">

 <plnk:role name="shippingService" portType="sif:shippingServicePT" />

 <plnk:role name="shippingServiceCustomer" portType="sif:shippingServiceCustomerPT" />

 </plnk:partnerLinkType>

</wsdl:definitions>

The partner link type shippingLT connects a service (shippingService) with its consumer (shippingServiceConsumer). In the WS-BPEL specification, the business process role is fixed service provider. Roles for port types are defined in shippingLT.wsdl and are represented as operations in shippingPT.wsdl.

The business process logic written is pseudo code is:

receive shipOrder

if condition shipComplete

 send shipNotice

else

 itemsShipped := 0

 while itemsShipped < itemsTotal

 itemsCount := opaque // non-deterministic assignment corresponding e.g. to

 // internal interaction with back-end system

 send shipNotice

 itemsShipped = itemsShipped + itemsCount

The process is instantiated when shipOrder is received. If this order has been executed then a shipNotice is replied. The situation is checked in the message header property shipComplete. Otherwise, a cycle is executed for order execution. At every step, part of the items are delivered and a notification is send. The counter is incremented with the number of sent items. The cycle exits when all items are delivered and then the process is terminated. In the abstract process, the number of delivered items at every step is non-deterministic. This information is retrieved from the back end system. From interactions point of view, process is very simple.

Initially, the process waits to receive an order message from a consumer and then replies with one or more messages. There are no error handlers, no compensators, no return values. There is no need for correlation sets coordination: when a new instance of the process is created, the process is restarted in parallel to wait for new order, and the current instance is executing the received yet order.

In CSP, the process is very simple as is shown below:

channel customer 0;

**shippingService**() = customer?shipOrder -> (**checkOrder**(shipOrder) |||

**shippingService**());

**checkOrder**(shipOrder) = (shipComplete -> customer!shipOrder -> **Skip**) []

(shipNotComplete -> **executeOrder**(shipOrder));

**executeOrder**(shipOrder) =

 (itemsShipped -> **Skip**) []

 (itemsNotShipped -> change\_itemsCount -> customer!shipOrder ->

**executeOrder**(shipOrder));

var count = 10;

**shippingServiceCustomer**() =

 if (count > 0) {customer!count -> {count--} -> **receive**()} else {**Skip**};

**receive**() = customer?shipNotice -> **receive**();

**System**() = **shippingServiceCustomer**() ||| **shippingService**();

#assert System() deadlockfree;

PAT, product for specification and verification of CSP models, is used here. In this specification, there is only one channel between service provider and service consumer. This channel is modelling the partner link from the WS-BPEL specification. The channel could be modelled with some capacity, but here only message can be exchanged through it, like in classic CSP.

The WS-BPEL process is modelled here as CSP process shippingService. This process initially is waiting to receive a shipping order through the channel. When the process receives an order, it starts its execution, but in parallel restarts a new its own copy to wait for a new order.

The subprocess checkOrder checks is the order has been executed yet. As result of this check, there are two possible events: the order is executed yet or not. With these two events is modelled the check in WS-BPEL process that is performed on the message text. The order is received as a parameter by the subprocess checkOrder.

If the order has been executed yet, the process sends through the channel a notification, which may be is the order in some other format. Otherwise, it starts the subprocess executeOrder. In this process, all manipulations with variables, messages and properties are abstracted to result events. If all order items have been sent then the process terminates. Otherwise, the back end systems is initiated. The last one sends information when some delivery is done. This is marked by an occurrence of change\_itemsCount. Through the channel, the consumer is informed about that delivery. Then follows a recursive execution of the subprocess with the last one items.

In this specification, instead of shippingNotice is returned shippingOrder. The idea is that a document-message like shippingOrder carry the state of business process and there is no need of other messages. It possible CSP process to use different message in that case, but this will not change the interaction flow.

In the CSP specification, there are consumer and system subprocesses. They are added for verification purposes of the whole system.

The main approach in this conversion from WS-BPEL to CSP is abstraction of data manipulations into events. Another, well known, approach is representation of cycle as recursive subprocess (subprogram). Conditional statements are modelled as choices among events. The simplifications can be done, because correlation sets are ignored in the model. They are used only by the consumer part. The process simply returns through the channel data (the order) that contain the dialog identifier.

# Conclusion

Attractive results in WSDL formalization with Z notation have not been achieved, because there are no behavior. WSDL specification of business process is simply interfaces to Web services.

On the other hand, WS-BPEL specification of business process specifies its behavior. Its formalization in CSP maximally abstracted many implementation details saving the original interaction flow. The CSP specification can then be formally verified. The CSP model is very compact and readable.

This example of formalization demonstrates an approach to formal verification of business processes. Full description of this research approach will be published in future.

**Acknowledgements**. Research in this paper are funded by Bulgarian Science Fund under contract ДФНИ-И01/12 “Modern programming languages, environments and technologies, and their application in education of software professionals”.

# References

1. OASIS, Web Services Business Process Execution Language Version 2.0, OASIS Standard, 11 April 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
2. W3C, Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl
3. ISO/IEC 13568:2002, Information technology -- Z formal specification notation -- Syntax, type system and semantics, http://www.iso.org/iso/home/store/catalogue\_tc/catalogue\_detail.htm?csnumber=21573