КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2015 Т. 7 № 3 С. 445–454

Ки&М

ПЛЕНАРНЫЕ ДОКЛАДЫ

УДК: 004.023

Deriving semantics from WS-BPEL specifications of parallel business processes on an example

V. Dimitrov

University of Sofia, Faculty of Mathematics and Informatics, Bulgaria, 1164 Sofia, 5 James Bourchier Blvd.

E-mail: cht@fmi.uni-sofia.bg

Received October 27, 2014

WS-BPEL is a widely accepted standard for specification of business distributed and parallel processes. This standard is a mismatch of algebraic and Petri net paradigms. Following that, it is easy to specify WS-BPEL business process with unwanted features. That is why the verification of WS-BPEL business processes is very important. The intent of this paper is to show some possibilities for conversion of a WS-BPEL processes into more formal specifications that can be verified. CSP and Z-notation are used as formal models. Z-notation is useful for specification of abstract data types. Web services can be viewed as a kind of abstract data types.

Keywords: parallel business processes, specification WS-BPEL, semantics

Извлечение семантики из спецификаций WS-BPEL обработки параллельных процессов в бизнесе на примере

В. Димитров

Университет Софии, Факультет Математики и Информатики, Болгария, 1164, г. София, б-р Джеймс Баучера, д. 5

WS-BPEL — это широко распространённый стандарт для спецификации распределенных и параллельных бизнес-процессов. Этот стандарт не подходит для алгебраических парадигм и парадигм направленных графов Петри. Исходя из этого, легко определить бизнес-процесс WS-BPEL с нежелательными особенностями. Именно поэтому проверка бизнес-процессов WS-BPEL очень важна. Цель этой статьи состоит в том, чтобы показать некоторые возможности для преобразования процессов WS-BPEL в более формальные спецификации, которые могут быть проверены. CSP и система обозначений Z используются как формальные модели. Система обозначений Z полезна для спецификации абстрактных типов данных. Web-сервисы могут рассматриваться как своего рода абстрактные типы данных.

Ключевые слова: параллельные бизнес-процессы, спецификация WS-BPEL, семантика

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”.

Citation: *Computer Research and Modeling*, 2015, vol. 7, no. 3, pp. 445–454.

Motivation

There are two kinds of business processes in WS-BPEL [OASIS…, 2007]: executable and abstract ones.

The behavioral semantics of executable business processes is well defined in WS-BPEL standard. There is only one problem with WS-BPEL extensions, because they go outside the notation framework; they are open and unpredictable, but without them the framework semantics is consistent.

The other category are the abstract business processes. Their intention is to describe Web services interactions without details on Web services internal implementations. The standard defines concretization procedure that can create an executable business process from an abstract one. But it is possible to generate with this procedure an executable business process with behavior different from that of the abstract one. It is possible, the executable business process to contain interactions that change the original ones, i.e. the executable process is not simply specialization of the abstract one. 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 verify abstract business processes because:

1. Verification of an abstract business process means a verification 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 is specified in two parts. First one is the WSDL [W3C…, 2001] specification of the Web services involved in the interaction. This specification includes the business process specification as a Web service. The second part is the WS-BPEL business process specification as Web services interactions. These both specifications are complementary because the business process, usually, is a Web service specified in WSDL. On the other hand, the business process as Web service is implemented in WS-BPEL. WSDL standard is extended to capture Web Services participating in the WS-BPEL business process and this is an essential part of WS-BPEL.

WSDL specifies only the interfaces and hides implementation details. WSDL specifications could be formalized. Why such a formalization is needed? WSDL is a XML based notation. Authors of XML argue that XML is readable for humans and computers. But XML specifications are verbose and not readable for humans. That is why, it is better, if Web services can be specified in some more compact notation, that is well better accepted by the humans. Such a tool is the Z notation [ISO/IEC 13568:2002]. Specifications in it tend to be very compact.

Z-notation is mainly used for specification of abstract data types, but it can be used for specification of algebras.

Web services can 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>
```

The application data schemas are imported in this part of the WSDL specification. Such a types here are shipOrder and shipNotice. They are the application data containers. Only the properties, defined on these messages, have impact on the message exchange among the business process Web services. These data types are modeled as basic types in Z notation:

$$[shipOrder, shipNotice]$$

Messages are modelled with Z schemas:

$$\begin{array}{|l} shippingRequestMsg \\ \hline shipOrder : shipOrder \\ \hline \end{array}$$

$$\begin{array}{|l} shippingNoticeMsg \\ \hline shipNotice : shipNotice \\ \hline \end{array}$$

Every message part in the Z-schemas is specified as a field with the same part name and the same type name.

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 the Z-notation, operations to be defined on the 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 is not very elegant approach for introducing the Web services, but it works.

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

$[WebService]$

$shippingServicePT \triangleq [ws: WebService]$

$shippingCustomerPT \triangleq [ws: WebService]$

shippingRequest

$\Delta shippingServicePT$

$input?: shippingRequestMsg$

shippingNotice

$\Delta shippingCustomerPT$

$input?: shippingNoticeMsg$

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:

$$itemCountType == 1..50$$

The properties are then represented as types:

$$shipOrderID == \mathbb{N}_1$$
$$shipComplete ::= False \mid True$$
$$itemsTotal == itemCountType$$
$$itemsCount == itemCountType$$

There are two deviations in the Z-notation schemas 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.

The aliases are properties placed on the messages. Here, they are modelled as functions from thr message type to the property type. There is no need to model XPath queries, because they are extensions to WS-BPEL. Queries written in other languages can be modelled in the same way. The aliases specification is more abstract:

$$
\begin{aligned}
&\mathit{shipOrderID\_shipOrder}: \mathit{shippingRequestMsg} \rightarrow \mathit{shipOrderID}\\
&\mathit{shipOrderID\_shippingNotice}: \mathit{shippingNoticeMsg} \rightarrow \mathit{shipOrderID}\\
&\mathit{shipComplete\_shipOrder}: \mathit{shippingNoticeMsg} \rightarrow \mathit{shipComplete}\\
&\mathit{itemsTotal\_shipOrder}: \mathit{shippingRequestMsg} \rightarrow \mathit{itemsTotal}\\
&\mathit{itemsCount\_shipOrder}: \mathit{shippingRequestMsg} \rightarrow \mathit{itemsCount}\\
&\mathit{itemsCount\_shippingNotice}: \mathit{shippingNoticeMsg} \rightarrow \mathit{itemsCount}
\end{aligned}
$$

One property could have many aliases with the same name. In the Z-notation, above defined functions are global ones and their names must be unique. So, alias name is formed by the property name and the 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 WS-BPEL business process.

Finally, as result of the modelling effort, the specification is very compact and very simple. It does not include the business process as a Web service. This specification could be used for Web Services development, but it is very simple without invariants.

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, the 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 taken 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 a fixed service provider. The roles of 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 a shipOrder is received. If the received order has been executed then a shipNotice is replied. This situation is checked in the message header property shipComplete. Otherwise, a cycle is started for the order execution. At every step, part of the items are delivered and a notification is send. The counter is incremented with the number of the sent items. The cycle exits when all items are delivered and only 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 backend system that actually register how many items have been send. From interactions point of view, this 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 can be 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 have some capacity, but here only a message can be exchanged through it, like in the classic CSP.

The WS-BPEL process is modelled as the 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 copy to wait for a new order.

The subprocess checkOrder checks the order. There are two possible events from the check: the order is executed yet or not. With these two events is modelled the check in the WS-BPEL process. 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 message in some 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 backend system 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 left items.

In this specification, instead of shippingNotice is returned shippingOrder. The idea is that a document-message, like shippingOrder, carries the business process state and no more other messages are needed. It possible, the CSP process to use different message in that case, but this would 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 abstraction of data manipulations into events is the main approach in this conversion from WS-BPEL to CSP. The representation of a cycle as a recursive subprocess (subprogram) is the other used approach. The conditional statements are modelled as choices among events. The correlation sets are ignored in the model, because they are used only in 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 for the modelled Web Services. The business process WSDL specification is simply an interface to several Web services.

On the other hand, the business process WS-BPEL specification specifies a behavior. Its formalization in CSP is maximally abstracted from 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.

References

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

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

W3C, Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl