A Pitfall with BPMN Execution

Christian Gutschier, Ralph Hoch, Hermann Kaindl, Roman Popp
Institute of Computer Technology, Vienna University of Technology
Gusshausstrasse 27-29, 1040 Vienna, Austria
{gutschier, hoch, kaindl, popp}@ict.tuwien.ac.at

***Abstract*—With its release 2.0, the language *Business Process Model and Notation* (BPMN) is supposed to be directly executable, without prior translation to another language such as *Business Process Execution Language* (BPEL). In fact, the XML definition of BPMN 2.0 allows for specifying executable models, by extending the graphical notation in this respect. However, we found a pitfall related to the number of parameters of *Service Tasks*. Even for very simple models, it is hard to make them fully compliant to the BPMN 2.0 standard. We propose a solution for circumventing this problem using wrappers. It leads, however, to dependencies on specific execution engines. While this problem may seem to be minor, it has certain wide-reaching consequences. In particular, it means that BPMN 2.0 cannot be used, in general, as a service orchestration language like BPEL.**

***Keywords*-BPMN, Service Task, Web Service**

I. Introduction

Business Process Model and Notation (BPMN) is originally a graphical language for visually defining business processes. [1] Its latest version 2.0 contains enhancements to the graphical notation, a metamodel, and XML specifications for making such models executable, when properly connected with Web services or Java code. More precisely, not every BPMN 2.0 is executable, but BPMN 2.0 allows specifying executable models.

We investigated how such models can actually be specified graphically with corresponding modeler tools, enhanced by (partly manually) provided XML specifications. In this course, we learned how to really execute such models, but we also found a specific pitfall. In fact, there is an explicit constraint — the BPMN 2.0 standard does *not* allow more than one input set and a single *Data Input* per *Service Task*. This restriction may seem small and unimportant at a first glance, but is has wide-reaching consequences.

As a running example, we use a very simple business process for a small company, including the tasks *Create Invoice*, and *Send Invoice*, of course in the given order. For each of these tasks we assume implementations as Web services for execution of this business process. The Invoice consists of two attributes — an address and an amount.

Using this simple example process, we illustrate the pitfall indicated above. We also try to remedy this problem with a few approaches based on *wrappers*. While they allow specifying models that are compliant to the BPMN 2.0 standard, their precise implementations are tool-specific. So, we also discuss a few execution frameworks in this regard.

The remainder of this paper is organized in the following manner. First, we present some background material in order to make this paper self-contained, and discuss related work. Then we elaborate on model definition and execution with BPMN 2.0, using our running example. The focus is on the pitfall that we found and our proposals for its remedy. Finally, we compare a selection of BPMN execution frameworks in this regard, and conclude.

II. Background and Related Work

BPMN is a graphical specification language, which provides symbols to model business processes, workflows and business activities. The first official BPMN version 1.0 was introduced and presented by OMG^{TM} (Object Management Group^{TM}) as a standard in 2006. Subsequent versions (1.1 and 1.2) provided only changes to model presentation and small corrections.

So, up to version 1.2, the main focus of the developers of BPMN was on the graphical representation of business processes. BPMN diagrams should serve the understandability of business processes for all parties involved. In particular, they should be understood both by IT and by business experts. Directly executing such business models was not possible, however.

The main approach for executing BPMN models up to version 1.2 was through their mapping to another language — Web Service Business Process Execution Language (WS-BPEL or BPEL, in short) [2], a block-oriented language. This approach has been discussed in [3], and in [4] an algorithm can be found for automatic translation of Business Process Diagram BPD (graph-oriented) components to a block-structured BPEL process. More precisely, three different approaches are suggested for this transformation. Well-structured BPD components can be directly mapped to BPEL structured activities. Non-well-structured but acyclic BPD components can be mapped to control link-based BPEL code. All remaining components can be mapped to event-action-rules. Unfortunately, there are some ambiguities in the specification of BPEL. These are discussed in greater detail in [5].

Only the next and current version (2.0) of BPMN [6], which was fully developed in 2011, brought important changes and interesting innovations. The most important innovations in version 2.0 were that the BPMN models could be stored in a standardized XML-based format, and the introduction of a metamodel. Based on that, exchange of BPMN 2.0 models between tools became possible, and direct execution of BPMN 2.0 models. So, mapping to another language for execution should no longer be needed according to [7], [8].

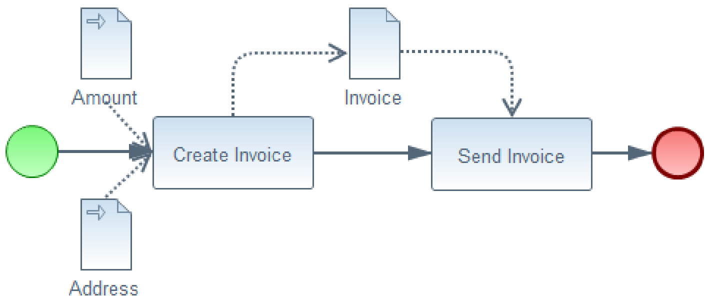

Figure 1: Basic BPMN 2.0 Model of Example Process

In addition, BPMN 2.0 allows the use of external specifications in a model via an import mechanism. Data structures that follow the XML standard can be used as reference structures. This is important for an automatic execution of business processes modeled in BPMN 2.0, since it enables the user to reference existing service implementations through a Web Service Description Language (WSDL) [9] file. The *Service Task* newly introduced in BPMN 2.0 specifies how WSDL references can be made and is thus the basis for automatic service execution.

The specification of WSDL allows interoperability between client applications and Web Services. To call operations of a Web Service, Simple Object Access Protocol (SOAP) [10] messages are exchanged between the participating actors. These messages are standardized as well and build a bridge between applications of different programming languages.

However, since the development in the Web Service area is so rapid and different requirements for Web Services can emerge, there are some varieties that the standard specifications of WSDL and SOAP permit. For example the structure of a WSDL file can be different depending on the SOAP binding style used. There are two different styles that are commonly used: RPC and document. The difference is how the messages that are passed between two actors are constructed. The RPC style is closely related to a normal method call in a programming language and involves a tighter coupling between WSDL and source code. The document style uses the possibilities of XML schema specifications to specify parameters and thus involves a looser coupling between services and source code. Furthermore, there are different ways to encode the messages, namely literal and encoded. All these differences influence the structure of the WSDL file as well as the format of the messages to be exchanged (SOAP). From all these possible combinations, only the document literal wrapped and RPC literal style are WS-I compliant (Web Service – Interoperability, an approach to further harmonize Web Service specifications). Since this is not the main focus of this paper, we refer the reader to [11], [12] for more detailed descriptions.

Since client applications require a Web service stub that handles the connection to the service implementation, these stubs have to be generated or created according to the WSDL specification of the Web service. There are several frameworks available that allow automatic generation of a client stub from a WSDL specification during runtime. These stubs are then used to create service calls and hide the marshalling of objects and messages that are exchanged between client and server. A call to a Web service then acts like a local call of a procedure.

Pillat et al. [13] discuss an extension of BPMN to help the user during the Software Development Process through the already available extensibility class defined in the BPMN 2.0 standard. However, the main problem addressed in our approach is not considered there.

III. BPMN 2.0 Model Definition and Execution

First, let us specify the simple process from our running example in BPMN 2.0, as shown in Figure 1. Based on it, we elaborate on its execution (based on Web services), which interestingly requires a change in the model. For addressing a pitfall involved that we found, we propose solutions using wrappers for parameter handling, in order to be compliant with the BPMN 2.0 standard.

A. *Specification of BPMN 2.0 Process Models*

The process of our running example has two tasks involved, one for the creation of an Invoice entity for two given attributes/inputs (an Amount of money and a recipient Address), and another one for sending out the created Invoice. These are simply modeled in Figure 1 as BPMN *Tasks*, which are connected with a line showing the control flow through the arrow head. The symbols for the data are connected with the *Task* symbols through dotted lines with arrow heads, visualizing data associations.

However, the model in Figure 1 does not specify any details on the implementation of the *Tasks*. Since BPMN 2.0 strives for combining views for business as well as technical users, an automatic execution of such a business process should be possible, but for this purpose the model needs to be made more specific. In particular, the general *Tasks* have to be replaced with *Service Tasks*, a specialization dedicated to execution of services. Since the BPMN 2.0 standard defines operating with Web services based on a WSDL description, we assume that both *Service Tasks* are implemented using common Web service technology.

The difference in the graphical notation is minor (as illustrated in Figure 3) but the implication of this substitution in the XML notation is rather important. There are several attributes

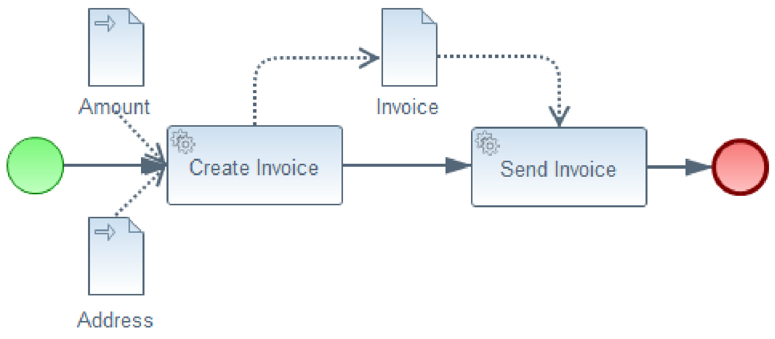

Figure 2: Adapted BPMN 2.0 Model of Example Process with *Service Tasks*

involved with calling Web services. Among other things, there are an *Input Output Specification* and associated *Data Input* and *Data Output Association*. This *Input Output Specification* defines the parameters for the Web service and may contain an additional mapping to local variables that are used in the BPMN process. The actual reference to the Web service is provided through an additional attribute of the *Service Task*, where an operation reference *operationRef* can be specified. This specification relates to a WSDL operation and is thus directly related to the Web service. Data structures can be imported from WSDL specifications and reused in BPMN. Together, these specifications allow for an automatic execution of *Service Tasks* that reference an existing WSDL specification.

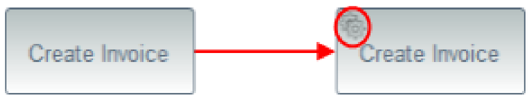

Figure 3: Substitute (General) *Task* with *Service Task*

Replacing the *Tasks* from Figure 1 with *Service Tasks* leads to an adapted model shown in Figure 2. This model is analogous to the one of the basic process shown in Figure 1, but it represents a business process with embedded service calls. These service specifications are not shown in the graphical representation of the process model, but only in the XML representation that the graphical notation is based on, as explained above.

So far, everything looks plausible and the process model more or less straight-forward. However, simply replacing the *Tasks* from Figure 1 with *Service Tasks* actually leads to a model that is *not* compliant with the BPMN 2.0 standard. This very standard imposes an additional constraint on *Service Tasks*, i.e., it does *not* allow more than one input set and a single *Data Input* per *Service Task*. Even this simple process model, however, has *two* input parameters for the *Service Task* implementing the creation of an Invoice.

The definition of the *Service Task* can be found in the official standard, where the following quote describes this only-one-parameter constraint in detail:

> ”The Service Task inherits the attributes and model associations of Activity (see Table 10.3). In addition the following constraints are introduced when the Service Task references an Operation: The Service Task has exactly one inputSet and at most one outputSet. It has a single Data Input with an ItemDefinition equivalent to the one defined by the Message referenced by the inMessageRef attribute of the associated Operation. If the Operation defines output Messages, the Service Task has a single Data Output that has an ItemDefinition equivalent to the one defined by the Message referenced by the outMessageRef attribute of the associated Operation.” [14, p. 158]

This constraint might be a reaction to Web Service specifications where the SOAP message only consists of one body part. However, the *Service Task* itself should not be involved with the actual encoding and message passing of a Web Service and should only deal with the parameters themselves. If the *Service Task* is supposed to receive an already fully constructed message object with all parameters embedded, it would make it difficult for somebody without technical background to understand how this service is invoked. Furthermore, the specification above does not only hinder invoking Web Services but also simple procedure calls in a programming language, which BPMN 2.0 is capable of. Since the encoding of parameters to message objects (in case of WSDL/SOAP) is not at the same level of abstraction as input parameters of methods/operations, this should not be mixed up. So, the definition of parameters of a *Service Task* should not involve any information about the actual encoding or how they are transported. This should be part of the *Service Task* and its specification (which BPMN already handles through an import statement and additional attributes).

In this paper, we simply refer to parameters that are passed to a method. For example, in the process shown in Figure 1 the *Task Create Invoice* has two input parameters *Amount* and *Address* that could be passed directly to a method implementation (for example: *public Invoice createInvoice(String amount, String address)*). The actual implementation is hidden from the BPMN 2.0 diagram and thus should also have no influence on the parameters themselves.

B. *Using a Wrapper for Parameter Handling*

The question is, if and how this problem with standard compliance caused by this constraint can be resolved. Our idea

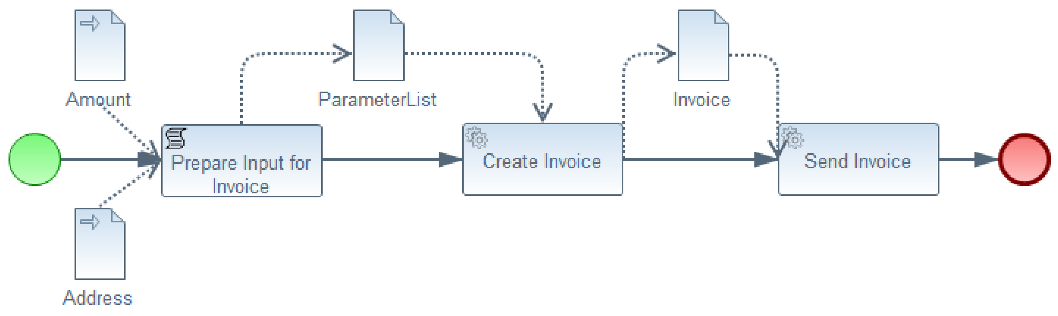

Figure 4: Modified BPMN 2.0 Model of Example Process with Additional Wrapper

was to use a wrapper for parameter handling.

There are several possibilities to accomplish this, all of which involve changes to the BPMN process as well. One possible solution is to change the WSDL specification of the Web service so that only one parameter is passed as input. This would make the service itself compatible with the BPMN *Service Task* specification, but would still require additional mapping techniques in the BPMN process to combine all inputs into one single input. Furthermore, it puts an uncommon constraint on the development of Web services, and only a small subset of available services can be integrated into business processes modeled in BPMN 2.0.

Figure 4 shows a modified process model for our running example, which uses a wrapper so that the *Service Task* does not directly have more than one input. In our example, a *Script Task* is used as the wrapper, because a *Script Task* can have, in contrast to a *Service Task*, more than one input set and more than one *Data Input*. The wrapper *Script Task* is added before the *Service Task*, and it receives all the inputs which the *Service Task* needs for its operation. The function of the *Script Task* is to prepare the inputs for the *Service Task* in such a way that they are combined into a single input. In this case, all inputs are combined into one map. Viewed from a higher level, however, this modified model is not really appropriate any more for a business expert, since preparing input for another *Task* is not really a business-relevant *Task* per se.

Figure 5 shows the XML representation of the *Script Task* used as a wrapper for the *Service Task Create Invoice*. Since both *Script Task* and *Service Task* are derived from the more general *Task* specification in the BPMN 2.0 metamodel, their syntax is similar. Both have an *Input Output Specification* and associated *Data Input* and *Data Output Association*. This *Input Output Specification* defines the parameters of the *Script Task* which are combined in a map variable. In contrast to a *Service Task*, a *Script Task* is designed for execution of simple tasks that are usually implemented in a scripting language. Therefore, it has an attribute *scriptFormat* instead of the attribute *operationRef*. This attribute specifies the language in which the code is written and which the process engine has to interpret and execute.

In this example, the language groovy is used as the script language. Furthermore, the *Script Task* has another important attribute called *Script*, in which the script itself is implemented. This is the main part of the *Script Task* and is used during execution. Figure 5 shows that in this attribute first an object of type “InvoiceParameterList” is created and stored in a variable “ParameterList”. The object “InvoiceParameterList” has two operations — one to add parameters and one to delete parameters. The behavior is similar to a Java HashMap, and is required to operate with WSDL specifications, since there simple or well-defined parameter types are allowed only. The two inputs of the *Script Task* are then added to the created object via the “addParameter” operation. In a last step, the object is set as the process variable that corresponds to the *Data Output* of the *Script Task*.

The approach described in Figure 5 shows how a complex object of type InvoiceParameterList is created and other parameters, which have been specified as the input of the *Script Task* PrepareParameters, are attached to it. After all parameters have been added in the “script” section, the resulting variable is set in the execution engine. Note, that the name of this resulting variable is set according to the output specification in the dataOutputAssociation. Thus the input and output of the *Script Task* can be specified according to the script in the “script” section, and the input/output behavior of the *Script Task* is fully specified. Since the script directly operates in the execution engine, it would also be possible to set other variables. Some frameworks automatically define an additional output specification for them.

However, this approach has a drawback. It requires the Web service to only take a single parameter, where all other parameters are embedded in one single object. A better solution would be to call the Web service directly from a *Script Task* or maybe an attached Java class. The idea is to hide the call to the Web service in a *Script Task* and thus avoid the unusual constraint with just one parameter. Figure 6 gives an example for the graphical notation for such a wrapper. As it shows, there is no indication for a Web service call given, which makes it harder to understand. Another major advantage is that this process model fits better the modeled process of our running example. A business expert can still see the essential tasks in this model and not an extra one irrelevant for the process. The specializations of *Task* to *Script Task* and to *Service Task*, respectively, are only visualized as small icons and should not be irritating to the business expert.

Another approach would be to use Java implementations

```
<!-- Script Task -->
  <scriptTask id="PrepareParameters" scriptFormat="groovy">
   <ioSpecification id="InputOutputSpecification_5">
        <dataInput id="dataInputScriptTaskAmount" itemSubjectRef="inputScriptTask"
            name="Amount"/>
        <dataInput id="dataInputScriptTaskAddress"
            itemSubjectRef="inputScriptTaskAddress" name="Address"/>
        <dataOutput id="dataOutputScriptTask" itemSubjectRef="outputScriptTask"
            name="Result"/>
          <inputSet id="InputSet_5">
            <dataInputRefs>dataInputScriptTaskAmount<dataInputRefs>
        <inputSet>
        <inputSet id="InputSet_9">
            <dataInputRefs>dataInputScriptTaskAddress<dataInputRefs>
        <inputSet>
        <outputSet id="OutputSet_5">
            <dataOutputRefs>dataOutputScriptTask<dataOutputRefs>
        <outputSet>
      <ioSpecification>
      <dataInputAssociation id="DataInputAssociation_8">
         <sourceRef>amount</sourceRef>
     <dataInputAssociation>
     <dataInputAssociation id="DataInputAssociation_10">
         <sourceRef>address<sourceRef>
     <dataInputAssociation>
     <dataOutputAssociation id="DataOutputAssociation_9">
         <targetRef>ParameterList<targetRef>
     <dataOutputAssociation>
     <script>

        def parameters = new
            at.ac.tuwien.ict.proreuse.webservices.icas.icasservices.InvoiceParameterList();
        parameters.addParameter("amount", amount);
        parameters.addParameter("address", address);

        execution.setVariable("ParameterList", parameters);
    <script>
  <scriptTask>
```

Figure 5: XML Representation of a *Script Task* Used as a Wrapper

as a back-end for *Script Tasks* and to employ them to call the Web services. Since Java classes are executed in the same runtime environment, they can access all available properties and values during the execution. In this case, the definition of the *Data Input* and *Data Output* could technically be omitted (which makes the overview of the business process difficult, of course). The BPMN 2.0 standard does not provide a clear specification on how other implementations can be attached to a *Script Task* or *Service Task*, but simply states that other implementations can be included. This means that the technical implementation of calling back-end Java classes is permitted, though not entirely specified. Using this method has the advantage that during execution there is full control on all variables and definitions but that the graphical notation does not fully represent the business process.

Both techniques, calling the Web service directly from a *Script Task* or a reference Java class, require to make manual Web service calls. This means that there needs to be an implementation available for the client part of the service call. This is in contrast to the normal *Service Task* where most frameworks support automatic generation of the required client stub from the referenced WSDL file. Since most frameworks already provide such a generation, their implementation can be reused. However, the call to create the client stub has to be provided manually. In addition, it is possible to use external frameworks to generate all necessary classes on the client side.

In addition, the BPMN 2.0 standard provides another possibility for mapping *DataObjects*, the *Assignment* construct in the *DataInputAssociation*. This construct allows mapping one *ItemDefinition* to another one and thus enables BPMN users to specify how locally defined *DataObjects* can be transferred to (parts of) input parameters. This is similar to what we accomplish with the *ScriptTask* but has the disadvantage that only short expressions can be used and that still the input parameter of a *ServiceTask* has to be defined as a more complex structure that embeds the local objects into one single object. Additionally, this approach requires the process designer to know of the existence and state of local *DataObject* definitions and thus the transparency of the process is obscured. Using active tool support, this approach has already been

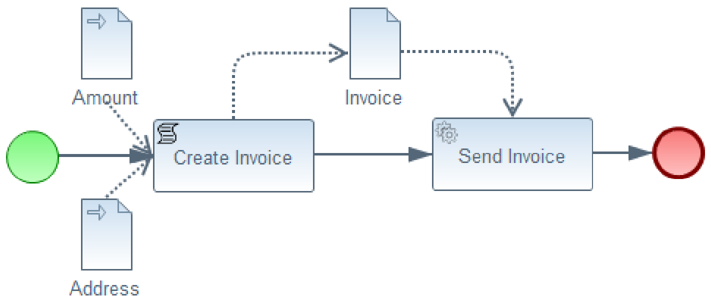

Figure 6: Modified BPMN 2.0 Model of Example Process with *Script Task* instead of *Service Task*

implemented in the BonitaSoft BPM framework [15], where custom connectors including custom types can be specified and automatically generated for further use (for more details see below in Section IV).

Since all these possible solutions rely on active tool support, it is necessary to state that the BPMN 2.0 standard does not allow for a compliant solution to this only-one-parameter problem it poses, so that solutions are tool-specific. We built and tested the discussed approaches primarily using the Activiti framework [16], and other frameworks may require different code elements (especially for the *Script Task*). However, the general idea behind the wrappers should work for all available frameworks. In addition, the frameworks provide the means to call the Web services and thus it might be necessary, depending on the framework, to provide client stub implementations or the shared objects in the same class path.

IV. BPMN Execution Frameworks

Since there are differences between BPMN execution frameworks, let us briefly compare a few of them, with a focus on the problem dealt with in this paper. Our running example was tested in the open source tools BonitaSoft BPM [15], Activiti [16], jBPM [17] and Camunda [18]. All four tools are based on the Java programming language and provide their own framework implementation of the BPMN 2.0 standard. While Activiti, jBPM and BonitaSoft BPM can integrate a Web service via the *Service Task* specification, Camunda can currently only integrate and use Java classes as reference structures in *Service Tasks*. However, as mentioned above, a Web service call can be accomplished through manual implementation in a Java class.

In Activiti, the stub classes for the Web service are automatically generated, while in jBPM it is necessary to manually create the stub classes for the Web service upfront. Our comparison of *Service Task* implementations in Activiti and jBPM revealed that there are several differences between them. For example, these frameworks handle the *Data Input* and *Data Output* specification of the *Service Task* differently. Furthermore, jBPM defines a specific name for the *Input Output Specification* of the *Service Task*, which must be “Parameter” for the *Data Input*, and “Result” for the *Data Output*. Otherwise the *Input Output Specification*, although correct according to the BPMN 2.0 standard, cannot be processed. Activiti allows the specification to use any identification name [16], [19].

Another difference regarding *Service Tasks* is that jBPM has implemented and enforces the only-one-parameter constraint of the BPMN 2.0 standard, while in Activiti is it possible to pass multiple parameters (which is not standard compliant). Also storing local variables is handled differently. While jBPM uses *Property* structures (which themselves reference an *ItemDefinition*) to store local variables, Activiti stores the local variables directly through an *ItemDefinition* [17].

BonitaSoft BPM handles *Data Input* and *Data Output* through the standard BPMN *Input Output Specification*. In contrast to the other tools investigated, BonitaSoft BPM uses a so-called connector implementation that allows the integration of external services. There are several predefined connectors for different purposes available (e.g., for Web Services and SOAP), but there is also the option to develop custom connectors for external services. These connectors are implemented in the Java programming language and can be exchanged between different processes defined in BonitaSoft BPM. This tool is compliant with the standard definition of BPMN 2.0 in this regard and, like jBPM, implements the only-one-parameter constraint on *Service Tasks*. However, to resolve the resulting restriction, a new custom type for each connector is introduced, where all parameters are encapsulated in one *ItemDefinition* that follows the custom type structure. This approach has the advantage that the constraint is not violated since only one input parameter is used, but if there is more than one parameter to be used in a *Service Task* (and thus in a connector) a mapping has to be specified. The idea is that local *DataObjects* from the process are mapped to parts of the input parameter for the *Service Task* using the BPMN 2.0 construct *Assignment* in the *DataInputAssociation*. Using expressions it is possible to access defined *DataObjects* and to map them to the corresponding parts in the custom type of the connector. This approach does not violate the constraint of BPMN 2.0 but has the disadvantage that it is not interchangeable with other execution engines since it heavily relies on the definition of custom types and connectors. Using the BonitaSoft BPM framework, the necessary specifications and external XSD definitions can be generated automatically and are interpreted during runtime. These additional pieces of information are not specified within the BPMN standard, however, and the tool has no option to express them graphically in the process diagram [15].

Since the *Script Task* is the main implementation vehicle for our wrappers, it is important how these scripts are processed. While most frameworks support only one script language, multiple languages are possible. Activiti, Camunda and BonitaSoft BPM, for example, use groovy, while jBPM specifies Java as script language. Both languages have similar syntax. However, it is important to note that there are minor differences in how variables are accessed and stored during runtime.

These differences make it hard to exchange BPMN business processes between these tools since all of them have their own small deviations from the BPMN 2.0 standard.

V. Conclusion

In this paper, we show with a simple business process example a pitfall during the execution of BPMN 2.0 business process models. The reason is an unusual constraint on the number of *Service Tasks* in the official language standard. In general, already given Web services take more than one parameter, and also for newly defined ones it would be very strange to limit them to taking one (or none). So, making use of existing Web services for executing BPMN 2.0 models is limited, and defining new ones with this constraint leads to strange interfaces. In fact, even process models and their data definitions suffer from this constraint, since their specification may not match the modeled processes well. Viewed from the perspective of Web services, this constraint prevents BPMN 2.0 from being used as an orchestration language.

Note, that all this is relevant only when caring about compliance to the official BPMN 2.0 standard. In fact, there are several BPMN execution frameworks available, and not all of them enforce this constraint imposed by the standard. However, what is such a standard good for, when only non-compliant models make sense in practice?

So, we studied potential standard-compliant solutions to this problem. They are based on the idea that some *wrapper* masks the passing of parameters to Web services or completely hides the call to the Web service. We present in this paper a few possibilities to accomplish this, first by utilizing *Script Tasks* to combine parameters into one complex object. Another solution could be to directly call Web services from *Script Tasks* or attached Java classes. However, since most of the frameworks have minor differences on how data is handled, all these solutions need to be adapted to a given framework. In effect, this means that a major intended advantage of standard compliance is not achieved, easy exchange of models between tools.

However, the question remains why this constraint has been introduced for *Service Tasks*. Since the *InputOutputSpecification* for general Tasks allows multiple *Input Sets*, this constraint is not necessary and restricts the usage of services in BPMN orchestration. A simple solution could be to just omit this constraint and leave the implementation of the BPMN standard open for multiple inputs. Additional information on the actual implementation (for Web Services this could involve the SOAP style or similar) could be provided by extending the interface definition with additional attributes. The mapping between Web Service parameters and local parameters could be established through the *DataInputAssociation*, which is specified in the BPMN 2.0 standard.

References

[1] OMG, Business Process Model and Notation (BPMN), Object Management Group Std., accessed: 2014-01-31. [Online]. Available: http://www.omg.org/spec/BPMN/

[2] OASIS. OASIS Web Services Business Process Execution Language (WSBPEL) TC. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel [retrieved: January, 2014]

[3] C. Ouyang, M. Dumas, A. H. M. Ter Hofstede, and W. M. P. Van der Aalst, "From BPMN Process Models to BPEL Web Services," in Web Services, 2006. ICWS '06. International Conference on, 2006, pp. 285–292.

[4] C. Ouyang, M. Dumas, W. M. P. V. D. Aalst, A. H. M. T. Hofstede, and J. Mendling, "From business process models to process-oriented software systems," ACM Trans. Softw. Eng. Methodol., vol. 19, no. 1, Aug. 2009, pp. 2:1–2:37. [Online]. Available: http://doi.acm.org/10.1145/1555392.1555395

[5] T. Hallwyl, F. Henglein, and T. Hildebrandt, "A standard-driven implementation of WS-BPEL 2.0," in Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10). New York, NY, USA: ACM, 2010, pp. 2472–2476. [Online]. Available: http://doi.acm.org/10.1145/1774088.1774599

[6] O. M. Group. Business Process Model and Notation (BPMN) Version 2.0. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/ [retrieved: 01, 2014]

[7] T. Allweyer, BPMN 2.0 - Introduction to the Standard for Business Process Modeling, 2nd ed. Books on Demand GmbH, Norderstedt, 2010.

[8] M. Chinosi and A. Trombetta, "BPMN: An introduction to the standard," Computer Standards & Interfaces, vol. 34, no. 1, 2012, pp. 124 – 134. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548911000766

[9] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) 1.1. [Online]. Available: http://www.w3.org/TR/wsdl [retrieved: January, 2014]

[10] World Wide Web Consortium (W3C). Simple Object Access Protocol (SOAP). [Online]. Available: http://www.w3.org/TR/soap/ [retrieved: February, 2014]

[11] M. Crasso, J. Rodriguez, A. Zunino, and M. Campo, "Revising WSDL documents: Why and how," Internet Computing, IEEE, vol. 14, no. 5, Sept 2010, pp. 48–56.

[12] C. Mateos, M. Crasso, A. Zunino, and J. Coscia, "Revising WSDL documents: Why and how, part 2," Internet Computing, IEEE, vol. 17, no. 5, Sept 2013, pp. 46–53.

[13] R. Pillat, T. Oliveira, and F. Fonseca, "Introducing software process tailoring to BPMN: BPMNt," in Software and System Process (ICSSP), 2012 International Conference on, 2012, pp. 58–62.

[14] OMG, Business Process Model and Notation (BPMN), Version 2.0, Object Management Group Std., Rev. 2.0, January 2011, accessed: 2014-01-31. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[15] Bonitasoft. Bonitasoft. [Online]. Available: http://www.bonitasoft.com/ [retrieved: February, 2014]

[16] Activiti. Activiti BPM Platform. [Online]. Available: http://www.activiti.org/ [retrieved: February, 2014]

[17] JBoss Community team. Jboss community jBPM. [Online]. Available: https://www.jboss.org/jbpm [retrieved: February, 2014]

[18] Camunda. Camunda BPM platform. [Online]. Available: http://www.camunda.com/ [retrieved: February, 2014]

[19] T. Rademakers, Activiti in Action: Executable business processes in BPMN 2.0, 1st ed. Shelter Island, NY: Manning Publications, 2012. [Online]. Available: http://www.worldcat.org/search?qt=worldcat_org_all&q=1617290122