BPMN Extensions for Decentralized Execution and Monitoring of Business Processes

Jonas Anseeuw, Gregory Van Seghbroeck, Bruno Volckaert and Filip De Turck
Department of Information Technology, Ghent University, B-9050 Ghent, Belgium
{jonas.anseeuw, gregory.vanseghbroeck, bruno.volckaert, filip.deturck}@intec.ugent.be

Keywords: Cloud Computing, Business Process Management, Monitoring, Business Process as a Service.

Abstract: Software-as-a-service (SaaS) providers are further expanding their offering by growing into the space of business process outsourcing (BPO). Therefore, the SaaS provider wants to administer and manage the business process steps according to a service level agreement. Outsourcing of business processes results in decentralized business workflows. However, current business process modeling languages, e.g. Business Process Execution Language (BPEL), Business Process Model and Notation (BPMN), are based highly on a centralized execution model and current BPMN engines offer limited constructs for federation and decentralized execution. To guarantee execution of business processes according to a service level agreement, different parties involved in a federated workflow must be able to inspect the state of external workflows. This requires advanced inspection interfaces and monitoring facilities. Current business process modeling languages must thus be extended to support monitoring in the specification, support modeling and support deployment of decentralized workflows. In this paper, correlation and monitoring extensions for BPMN are described. These extensions to BPMN are described such that the existing specification can still be used as is in a backwards compatible way.

1 INTRODUCTION

Software-as-a-service oriented software companies want to add value to their offering by providing systematic and controlled delegation of many of the steps of a company's business process, also known as business process outsourcing (BPO). The companies thus administer and manage the business process steps according to a service level agreement. SLA can be an agreement on QoS parameters, but this is also an agreement on the different interfaces between partners.

For example, the process of designing a product needs both business activities and simulation activities (often in an interleaved sequence). Business activities represent mainly data in and output (e.g. constraints, design parameters, etc.) tasks. Simulation activities take models and parameters as input, and analyze usually characteristics that are mentioned in the user-defined constraints and that must be met. Business activities form a high-level view on the process with branches, loops and concurrent activities, referred as the *business workflow*. Simulation activities are usually needed in the course of a business workflow execution in order to validate design parameters early, referred as the *simulation workflow*. Figure 1 shows business and simulation workflows and their relation to each other. The simulation workflows are pluggable in the business workflow. As depicted in figure 1, Company A has control over the business workflow (including the simulation workflows from Company B and C).

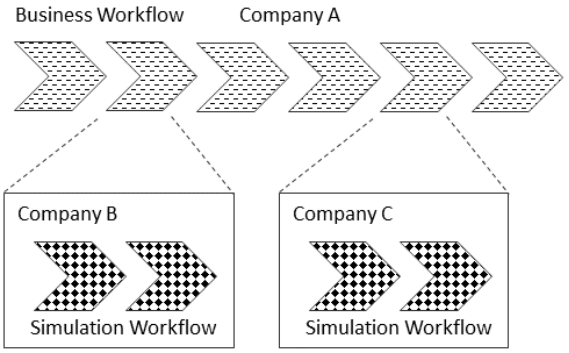

Figure 1: Relation of Business- and Simulation Workflows.

BPO results in decentralized business process flows: both inter-company flows, i.e. federated business process flows that cross the borders of companies, as well as intra-company workflows, distributed

business flows within the datacenters of one company. The business process modeling languages constructs offered by the current generation of workflow languages, e.g. Business Process Execution Language (BPEL) (OASIS, 2007) or Business Process Model and Notation (BPMN) ((OMG), 2011), are based highly on a centralized execution model and current BPMN engines offer no constructs for federation and decentralized execution.

The different parties involved in a federated workflow must be able to inspect the state of the external workflows, all the while hiding workflow implementation details. Business Process Modeling (BPM) languages must thus be extended to support modeling, monitoring and deployment of decentralized workflows in the specification.

The approach described in this paper is to extend BPMN using its extensibility such that the extensions are backwards compatible with the existing BPMN specification. The remainder of this paper is structured as follows. In Section 2, relevant related work is discussed. Sections 3 and 4 handle advanced correlation and monitoring extensions in BPMN respectively. Finally, our conclusions are drawn and future work is discussed in Section 5.

2 RELATED WORK

Previous research (Van Seghbroeck et al., 2007)(Stefanescu et al., 2014)(Barros, 2015)(Dumas and Kohlborn, 2015) has come to the same result with regard to the different steps in the development cycle of decentralized workflows or choreographies depicted in Figure 2. First, a top-down approach describes the service choreography, which can be validated. When the choreography is described, and after validation, all the different participants' stubs are extracted. As a result of the stub extraction and the different implementations, it is possible to use common process engines to execute a choreography.

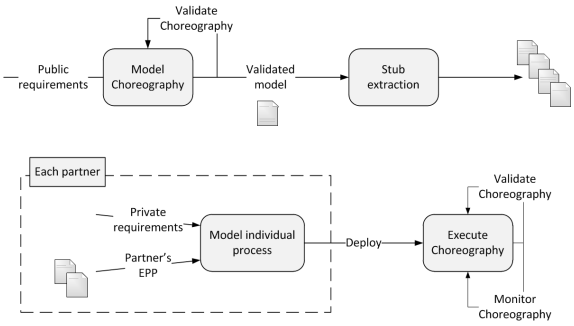

Figure 2: The complete development cycle for service choreographies.

The European Project CHOReOS (Large Scale Choreographies for the Future Internet) (Autili et al., 2014), which ended in 2013, resulted in a very interesting implementation of this development cycle. CHOReOS focusses only on BPMN, to describe its choreography, and to describe and execute the different participants parts of the choreography. CHOReOS, unfortunately, only monitors system level KPIs. In the monitoring model described in this paper, it is also possible to monitor application specific data.

The JBoss Savara[1] project can be used to create service choreographies. It uses Web Service Choreography Description Language (WS-CDL) (Kavantzas et al., 2005) to describe the choreography, but only some basic tools to monitor and validate the choreography are available. The Eclipse BPMN Modeler[2] can only be used to design choreographies, it does not have an execution environment incorporated, nor are there monitoring tools.

Correlating messages in choreographies comes with another level of complexity, not only do messages have to be correlated to each participant's workflow instance, but they also have to correlate these individual participants to the overall choreography instance. To the authors' knowledge, there is only one specification, WS-CDL, which has a very elaborate definition of and view on correlation. Since BPMN does not offer this, extensions to BPMN are needed.

Since there are no clear choreography execution environments yet, there are no ready-to-use monitoring environments to monitor choreography workflows. Monitoring, using the current tools and frameworks, would entail consolidating and aggregating the monitoring info from all the different choreography participants. It is clear that this is a fairly impossible task since every workflow engine has its own way to store the monitored data and has their own APIs to reference this information. The lack of a standardized monitoring API is a real problem here. Research has dealt with monitoring choreographies (Wetzstein et al., 2010; Lazovik et al., 2004; Roder et al., 2011), but very little research has been performed aimed specifically at BPMN. Another way to monitor a distributed environment is having a centralized system in place (e.g. Business Activity Monitoring) that gathers all necessary information from the individual partner's workflow engines, either via a pull or push method, or via monitoring agents. Another problem with regards to monitoring is that most of the current engines only provide monitoring information about system-wide aspects (e.g. number of re-

[1] http://savara.jboss.org

[2] http://eclipse.org/bpmn2-modeler

quests) or about aspects related to particular processes or particular process instances (e.g. execution time, a log trace for the different activities of a process). Adding monitoring points as part of a specific process or even application is not possible yet. BPMN is the only standardized specification that already supports including monitoring injection points with its monitoring and auditing element. However, except for describing extensible placeholders, the specification claims details are out of scope and are left to the implementing BPMN engines.

3 CORRELATION EXTENSIONS

As mentioned in related work, BPMN doesn't provide enough functionality to correlate messages in choreographies. BPMN already has some notion of correlation. This is done in the *Collaboration* definition, more specifically in its *Conversations*. A *Conversation* is defined by an array of *CorrelationKeys*. The BPMN specification claims that the *CorrelationKeys* can be used to tie messages to a specific process instance. Correlating messages to process instances may be enough to support centralized workflows, but this is insufficient for decentralized workflows. In a decentralized context, it is key to also uniquely and formally define correlations between all process instances of all parties involved in the decentralized workflow. This requires thorough knowledge on how all instances are created and on how those instances can be correlated via the messages communicated between them. Such a complex model is possible in WS-CDL, by means of its *Identity* types, but not in BPMN. There is another big difference between correlation in BPMN and WS-CDL, WS-CDL defines correlation on *ChannelTypes*, well-defined communication endpoints of a *Participant*, whereas in BPMN it is all done on the *Conversations*, which define the communication potential between two or more *Participants*. There are two possible approaches to improve correlation in BPMN:

1. Extend the BPMN Correlation mechanism to hold similar information as the WS-CDL mechanism.
2. Aid the users by improving how correlation is perceived in BPMN, i.e. add semantical meaning to particular constructs and combinations of conversations.

3.1 Extending BPMN

The first option results in less conversations and most resembles the usage of *ChannelTypes* and *Identities* in WS-CDL. A BPMN *Conversation* can be interpreted as a WS-CDL *ChannelType*. WS-CDL has four distinct correlation types: primary, alternate, derived and association. With these four identity types, it is possible to create complex correlation relations. BPMN's extension mechanism can be used to add an attribute to the *CorrelationKey* element. For example the attribute type, which can have the following values: primary, alternate, derived, or association. This way, we can mimic WS-CDL correlation types in BPMN.

3.2 Semantical Meaning

In the second option, the user interface will visualize correlation between messages or between conversations different. The user only has to be aware whether *CorrelationKeys* are used to correlate messages or conversations. *Message CorrelationKeys* in BPMN resemble the *Primary* and *Alternate identities* from WS-CDL. They are used to correlate messages belonging to the same *Conversation*. *Conversation CorrelationKeys* on the other hand are used to correlate conversations. *Derived* and *Association* identities take up this role in WS-CDL.

4 MONITORING EXTENSIONS

BPMN already has two placeholders available, auditing and monitoring, but as the specification mentions the actual definition of auditing/monitoring is out of scope of the specification. It is up to the implementers to specify how these elements are used and extended. Auditing and monitoring can be defined for all *FlowElements* (e.g. *Activities*, *Gateways*, *Events*, *Data Objects*, *Data Associations* and *Sequence Flows*), with the only limitation, that these *FlowElements* have to be part of a *Process* flow. Auditing and monitoring can also be set for a *Process* itself, which can be used (for instance) to define process-wide actions.

Setting the audit and monitoring tags for each *FlowElement* would be too time-consuming and prone to mistakes. Besides that, in many cases generic monitoring statements (e.g. *log servicetime between TaskA and TaskB*) can be used to define monitoring. Consequently, it would be more interesting to define the monitoring in a separate declarative model (as an extension on BPMN). References to elements in this new model can be used to connect the *FlowElements* and the *Process* to specific monitoring points. These monitoring points are described in declarative rules.

The following subsections describe in more details the declarative monitoring model, monitoring in-

formation, declaration, visualization and implementation of monitoring.

4.1 Monitoring Model

Several options are available for the Monitoring Model. Either a procedural model or a declarative model. Using a declarative model allows to describe the behaviour of the monitoring framework. Instead of describing in detail when to place a monitoring point, the model can describe under which conditions the framework should place a monitoring point. It is with a declarative model also possible to have simple rules that can result in multiple monitoring points. Some examples:

- after each Activity, log the name of the Activity
- servicetime between TaskB and TaskG
- delay between PartnerA and PartnerB

4.2 Monitoring Information

Metadata

The monitoring metadata describes which data can be captured from the *FlowElements*.

local_time
The timestamp as set by the participants system. In the assumption that all servers involved in the execution of one particular participant's process are in sync.

correlation_keys
An array of all the instances of the correlation keys as defined by the BPMN model.

process_id
The unique identifier for this process as set by the participant's system.

task_id
The unique identifier for this task as defined in the BPMN model.

data_context [optional]
An optional array of all data currently used in this process' instance. This is used to monitor application specific data and is realised by using *Data Objects*.

4.3 Describing Monitoring Declarations

There are different types of monitoring points: single monitoring points, monitoring ranges and monitoring aggregates. Single monitoring points are associated with a specific or all *FlowElements*. This can be a single *Activity* (e.g. *Task*, *Sub-Process*, *Call Activity*), *Events*, etc. Monitoring ranges are used to monitor data that involves multiple *FlowElements* (e.g. monitoring the service time between two *FlowElements*). A monitoring range implies that multiple single monitoring points are set. Monitoring aggregates are similar to monitoring ranges, but they aggregate data within a certain scope (e.g. over all process instances).

Single Monitoring Points

Single monitoring points can be set before, after or during either all *FlowElements* or a specific *FlowElement*. A *FlowElement* can be specified by a BPMN attribute (e.g. *id*). The second function parameter ω determines where the monitoring point should be set. Setting monitoring points results in capturing all data specified by the metadata as specified in section 4.2. In order to additionally monitor application specific data, *Data Objects* can be associated with monitoring points.

$$\texttt{log}(FlowElement[:id],\omega) \equiv \{local\_time,...,data\_context\} \quad (1)$$

Monitoring points can be placed depending on the outcome of a function. For example, function e can be $<$, $>$, $=$, *and*, *or*, *not*, etc. x and y can be *FlowElements*.

$$\texttt{e}(x,y,\omega) \quad \rightarrow \quad \texttt{log}(FlowElement,\omega) \quad (2)$$

Monitoring Ranges

Monitoring ranges are associated with multiple *FlowElements*. Therefore an ordered list of *FlowElements* is passed to function f.

$$\texttt{f}(FlowElement,\dots) \quad (3)$$

An example is monitoring the servicetime between two *FlowElements*. The function *servicetime* implies that a single monitoring point must be placed after *FlowElementA* and before *FlowElementB*.

$$\begin{aligned}&\texttt{servicetime}(FlowElementA, FlowElementB)\\ &\quad\rightarrow \texttt{log}(FlowElementA, after)\ as\ x\\ &\quad\wedge \texttt{log}(FlowElementB, before)\ as\ y\\ &\quad\wedge y.localtime - x.localtime\end{aligned}$$

Monitoring Aggregates

Monitoring aggregates are similar to ranges, but they aggregate all monitoring data within a scope (e.g. average, sum, count, etc.). Parameter α can be function (1),(2) or (3).

$$g(\alpha, scope) \quad (4)$$

The monitoring scope can be:

- process instance, e.g. count of all tasks within a specific process instance.
- process: range of process instances, e.g. count of all tasks over a range of process instances.
- system: range of processes, e.g. when you want to know the service time of a particular participant over different processes.

4.4 Visualization

In order to allow non-developers (business users) to specify where to place monitoring points, a more visual representation of monitoring points is discussed in the next subsections.

Single Monitoring Points

Figure 3 corresponds to the first rule (1). A monitoring point can contain a *Data Object* for application data to be logged.

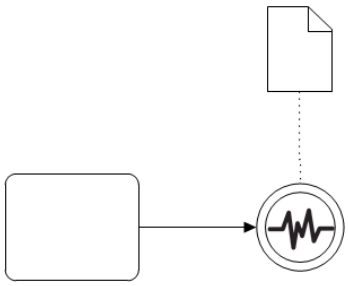

Figure 3: Placement of a single monitoring point.

Figure 4 corresponds to the second rule (2). For example a monitoring point will be placed if a *Data Object* and a particular *Message Event* have occurred.

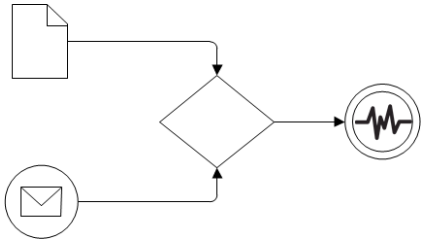

Figure 4: Placement of a conditional single monitoring point.

Monitoring Ranges

Figure 5 corresponds to the third rule (3).

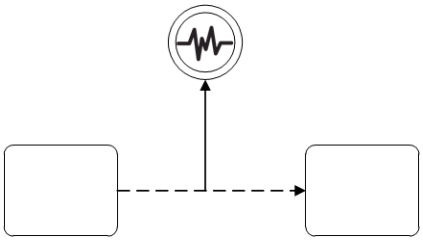

Figure 5: Placement of a monitoring range.

4.5 Implementation

Since BPMN is persisted in XML, the declarative rules should also have an XML counterpart, e.g. RuleML (a markup language for rules). Implementation of monitoring in BPMN can then be achieved by intercepting messages between tasks or by adding a wrapper around tasks. Automatically adding a wrapper around the monitoring tasks allows for more advanced monitoring capabilities. Figure 6 shows this wrapper.

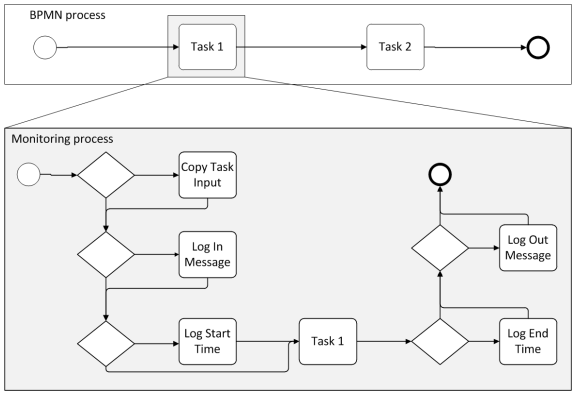

Figure 6: Wrapper around tasks to enable monitoring.

All of the supported monitoring operations must be included in the process. Whether or not they are active depends on the decisions in the process.

5 CONCLUSIONS

In this paper a number of backwards compatible extensions to the BPMN business process modeling language are presented to support correlation and monitoring in decentralized workflows. Correlation in decentralized workflows is achieved by using BPMN's extensibility. The BPMN correlation mechanism can than hold similar information as the WS-CDL mechanism. Another option is to aid the user by improving how correlation is perceived in BPMN. BPMN already has two placeholders available, auditing and

monitoring, but it is up to the implementers of the BPMN engine to specify how these elements are used and extended. Therefore, a declarative monitoring model as an extension on BPMN is described. This model results in declarative rules. Since BPMN is persisted in XML, the declarative rules also have an XML counterpart, e.g. RuleML. These rules have also a notation model. Finally, implementation of the monitoring model can be achieved by adding a wrapper around tasks.

Further research will show which correlation extension option is the more favorable. In future work tooling will be developed allowing non-developers (business users) to design and deploy decentralized business processes. A monitoring API will be designed to monitor these decentralized business processes.

ACKNOWLEDGEMENTS

The iMinds D-BASE project is co funded by iMinds (Interdisciplinary Institute for Technology), a research institute founded by the Flemish Government with project support of the IWT.

REFERENCES

Autili, M., Inverardi, P., and Tivoli, M. (2014). Choreos: Large scale choreographies for the future internet. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, pages 391–394.

Barros, A. (2015). Process choreography modelling. In vom Brocke, J. and Rosemann, M., editors, *Handbook on Business Process Management 1*, International Handbooks on Information Systems, pages 279–300. Springer Berlin Heidelberg.

Dumas, M. and Kohlborn, T. (2015). From business process models to service interfaces. In vom Brocke, J. and Rosemann, M., editors, *Handbook on Business Process Management 1*, International Handbooks on Information Systems, pages 557–578. Springer Berlin Heidelberg.

Kavantzas, N., Fletcher, T., Burdett, D., Lafon, Y., Barreto, C., and Ritzinger, G. (2005). Web services choreography description language version 1.0. Candidate recommendation, W3C. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.

Lazovik, A., Aiello, M., and Papazoglou, M. (2004). Associating assertions with business processes and monitoring their execution. In *Proceedings of the 2Nd International Conference on Service Oriented Computing*, ICSOC '04, pages 94–104, New York, NY, USA. ACM.

OASIS (2007). OASIS Web Services Business Process Execution Language. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

(OMG), O. M. G. (2011). Business process model and notation (bpmn) version 2.0. Technical report.

Roder, A., Lehmann, M., and Kabitzsch, K. (2011). Monitoring service choreographies. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 142–147.

Stefanescu, A., Wieczorek, S., and Schur, M. (2014). Message choreography modeling. *Software & Systems Modeling*, 13(1):9–33.

Van Seghbroeck, G., De Turck, F., Dhoedt, B., and Demeester, P. (2007). Web service choreography conformance verification in m2m systems through the pix-model. In *Pervasive Services, IEEE International Conference on*, pages 385–390.

Wetzstein, B., Karastoyanova, D., Kopp, O., Leymann, F., and Zwink, D. (2010). Cross-organizational process monitoring based on service choreographies. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2485–2490, New York, NY, USA. ACM.