Cross-organizational Process Monitoring based on Service Choreographies

Branimir Wetzstein, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, and Daniel Zwink

Institute of Architecture of Application Systems, University of Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

Branimir Wetzstein, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, and Daniel Zwink. 2010. Cross-organizational process monitoring based on service choreographies. In *Proceedings of the 2010 ACM Symposium on Applied Computing* (SAC '10). ACM, New York, NY, USA, 2485-2490. DOI=10.1145/1774088.1774601 http://doi.acm.org/10.1145/1774088.1774601

BIBTEX:

```
@inproceedings{Mon4Chor,
  author    = {Branimir Wetzstein and others},
  title     = {Cross-organizational Process Monitoring
               Based on Service Choreographies},
  booktitle = {2010 ACM Symposium on Applied Computing (SAC '10)},
  year      = {2010},
  pages     = {2485--2490},
  publisher = {ACM},
  doi       = {10.1145/1774088.1774601}
}
```

Cross-Organizational Process Monitoring based on Service Choreographies

Branimir Wetzstein, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, Daniel Zwink
Institute of Architecture of Application Systems
Universitaetsstr. 38
Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

ABSTRACT

Business process monitoring in the area of service oriented computing is typically performed using business activity monitoring technology in an intra-organizational setting. Due to outsourcing and the increasing need for companies to work together to meet their joint customer demands, there is a need for monitoring of business processes across organizational boundaries. Thereby, partners in a choreography have to exchange monitoring data, in order to enable process tracking and evaluation of process metrics. In this paper, we describe an event-based monitoring approach based on BPEL4Chor service choreography descriptions. We show how to define monitoring agreements specifying events each partner in the choreography has to provide. We distinguish between resource events and complex events for calculation of process metrics using complex event processing technology. We present our implementation and evaluate the concepts based on a scenario.

Keywords

Business Activity Monitoring, Cross-Organizational Monitoring, Service Choreography

1. INTRODUCTION

Business Process Management (BPM) encompasses methods, techniques, and tools that allow organizing, executing, and measuring the processes of an organization [12]. When BPM is layered over a Service Oriented Architecture (SOA) [9], services are used for implementing activities of business processes. In the context of SOA, business processes are modeled and executed using the (WS-)BPEL language, which is a workflow language for orchestration of Web services. While a service orchestration implements an executable private process model implemented by a single participant, a service choreography models the publicly visible processes and message exchanges between participants from a global viewpoint [10]. BPEL4Chor is a BPEL extension for modeling service choreographies [3].

For controlling the achievement of business goals especially in business processes and measuring process performance, business activity monitoring (BAM) technology enables continuous, near real-time event-based monitoring of business processes based on key business metrics, also known as key performance indicators (KPI) [11]. Business process monitoring has been traditionally focused on intra-enterprise processes. Today, companies are forced to collaborate in a more open manner in order to meet joint customers' needs. There is also more and more outsourcing of parts of business processes to external companies. Thereby, an intra-enterprise business process is fragmented into a cross-organizational process and the source company is often still interested in monitoring of the outsourced process fragment. A well-known example is shipment tracking whereby the shipper opens its process to some extent to the customer. Thus, there is a need for companies to interchange monitoring data of their business processes with other companies.

In this paper we present a solution to this problem by describing an event-based approach to cross-organizational monitoring based on service choreography descriptions. We use BPEL4Chor choreographies as basis for specification of so called monitoring agreements. A monitoring agreement specifies which events need to be provided by each partner in the choreography for building a monitoring solution. In particular, we distinguish between resource events which are defined based on the abstract processes in the choreography description and complex events needed for calculating higher-level process metrics using a complex event processing (CEP) language [8]. In order to support event correlation across partners in a choreography, we show the need for a choreography instance identifier and describe how it can be used in SOAP-based communication. We have implemented the approach in the Web services setting by extending an existing BPEL engine and using a CEP framework.

The rest of the paper is organized as follows. In Section 2 we present the motivation for our work based on a scenario which we use in the rest of the paper to present examples for our concepts. In Section 3 we depict the overall approach. Section 4 describes in detail how monitoring agreements are modeled. Section 5 deals with the monitoring infrastructure and event correlation in choreographies. In Section 6 we present related work, and finally, in Section 7 we conclude the paper and outline our future work.

2. SCENARIO AND MOTIVATION

For explaining the motivation and concepts of our work we have chosen a purchase order scenario as illustrated in the BPMN diagram shown in Figure 1. The diagram shows a chorography between a customer, a reseller, and a shipper

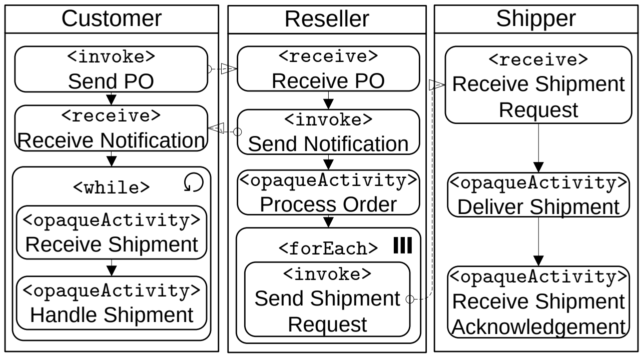

Figure 1: Purchase Order Scenario

(other involved participants such as suppliers have been omitted for space reasons). The customer sends an order request with details about the required products and needed amounts to the reseller. The reseller confirms the order by sending a notification to the customer. The reseller processes the order by ordering products from suppliers if needed, packages them and notifies the shipper. The order can be split in several parts if some of its parts take much longer to deliver. In that case the order parts are shipped separately. The shipper delivers the products to the customer.

Firstly, note that the diagram shows only public processes of the participants and their interactions. Private parts of the process are denoted by *opaque* activities and their implementation is not exposed to other participants. Note also that for this paper we assume that the partners have agreed on this choreography using the BPEL4Chor language [3]. A BPEL4Chor choreography description consists of a set of abstract BPEL process models (in our scenario three of them), one for each participant type (customer, reseller, and shipper), and a topology document which specifies how these abstract processes are connected together by message links. The choreography description also specifies concrete participants for participant types (e.g., two concrete shipping companies or concrete customers involved in the choreography; BPEL4Chor also supports dynamic sets of participants; however their monitoring is out of scope of this paper and part of our future work). Each partner implements its abstract business process as defined in the choreography locally typically using WS-BPEL, but not necessarily so. A partner could also use, for example, Java as long as it behaves according to the specified abstract BPEL process [4].

In the following we will motivate the need for our approach based on scenario examples. Considering the monitored objects, i.e., what is to be monitored, we can distinguish between process tracking and evaluation of process metrics. In process tracking, partners want to track the state of the choreography beyond their own process. In our scenario, for example, the customer is interested in tracking how far the order processing is. Obviously, this information can be provided by the reseller and shipper when they publish events as their process is executed, such as `Order received`, `Order processed`, `Shipment request received` and so on. We assume in this paper, that partners are willing to provide this information (or a subset of it) as long as it is part of its "public" process as modeled in the choreography. Note that we assume here that there is no special Web service operation provided by the reseller and shipper for inquiring this information; this would be a special case, and we concentrate on event-based monitoring in this paper. For process tracking, one has to agree for *which* process resource and which state change of that resource the event is to be published, *what* data (process data and IDs for correlation) is transmitted in the event, and *where* the event can be retrieved (on which messaging queue or pub/sub topic). Obviously, for unambiguous specification of the *which* and *what* question, an underlying choreography model is needed as basis. This is in our case the BPEL4Chor description. Process tracking relies only on state changes of process resources (in case of event-based monitoring also known as resource events). Besides process tracking it is often needed to evaluate metrics based on complex events. These metrics are then used e.g. as basis for definition of Service Level Agreements (SLAs) or Key Performance Indicators (KPIs). Consider, for example, the metric *order fulfillment lead time* which could be measured in our scenario from the start of the activity `Receive PO` in the reseller process until the `While Loop` completes in the customer process. Therefore, corresponding events have to be gathered, correlated and their timestamps subtracted. In general, thus we have to be able to specify CEP-like complex events based on events of different partners. The problem which arises here is that of event correlation in the choreography. In particular in this scenario we have to be able to correlate process instances within a choreography instance execution. In Section 5 we will explain in more detail, why in the general case, a special technical choreography instance ID is needed which has to be transported on protocol level, e.g., as part of SOAP headers.

In cross-organizational monitoring, obviously there are privacy issues. Firstly, the assumption of our approach is that partners are willing to provide only monitoring information on their public processes, but not private processes. This is why we have chosen to take a choreography description as basis for monitoring specification. It should also be possible to specify events selectively even for the public process. One could think of different monitoring levels dependent on how much the customer wants to pay for that information (if we assume that monitorability is part of service levels and is sold as a feature). Another issue is to be able to selectively restrict which partners can see which events.

3. OVERVIEW OF THE APPROACH

Figure 2 sketches the main concepts of our approach. The service choreography description can be seen as an agreement between partners on their public processes and message exchanges. We base our approach on a BPEL4Chor service description in which each partner exposes an abstract BPEL process [3]. We introduce a *monitoring agreement* which is an XML-based document specifying monitoring aspects between partners based on the choreography description. A monitoring agreement consists of a set of *resource event* definitions and *complex event* definitions. Resource events are defined based on abstract BPEL processes in the choreography by specifying at which BPEL resource and for which state of that resource an event is to published, which data it should contain, and where it should be published (at which message queue or pub/sub topic). Complex events are defined based on resource events and other complex events using a Complex Event Processing (CEP) language. They are needed for calculating process metrics. Both resource events and complex events are exchanged between partners over *message queues* or alternatively *pub/sub topics*.

Considering the methodology in creating corresponding monitoring agreements, there are two possible approaches. In a top-down approach the parties agree on what is to be monitored and create together a monitoring agreement document, possibly during creation of the choreography document itself. The document is then deployed to each party's infrastructure. The infrastructure is configured considering which events it has to publish to other partners and which events it retrieves from others. A more dynamic, bottom-up approach would imply that each partner creates the corresponding XML document (which is not yet an "agreement") independently of other partners and specifies which events it provides (and optionally also which events it requests) to partners in the choreography, possibly exposing different monitoring levels based on e.g. service levels and price the requester wants to pay. This monitoring document could then be published to a service registry together with the WSDL and choreography document. Obviously, in such a scenario a matchmaking phase is needed which checks whether requested and provided monitoring events match finally creating a *monitoring agreement* as a result. In this paper we focus on the top-down approach and leave the bottom-up one for future work.

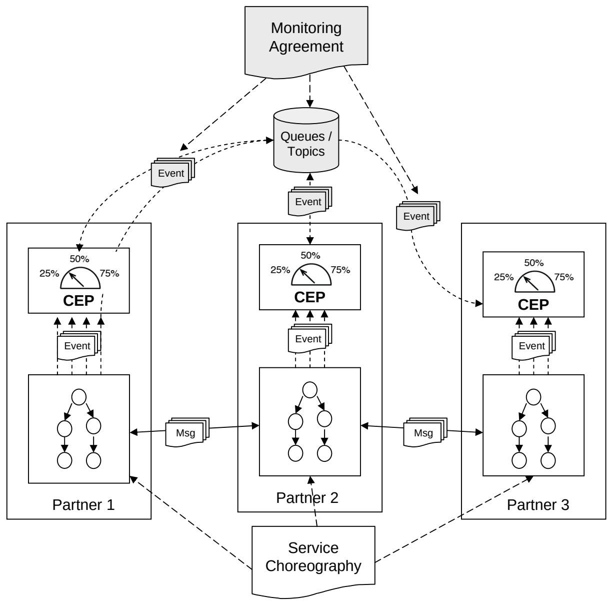

Figure 2: Overview of the Approach

Considering the lifecycle in creation of corresponding cross-organizational monitoring solutions we can distinguish between three phases: creation of monitoring agreements, deployment, and the concrete monitoring. After creation of the monitoring agreement document, it is used by each partner to configure its middleware for monitoring in the deployment phase. That involves configuring its own process middleware eventing infrastructure for publishing events to the specified destinations and subscribing to event queues or topics for getting events from other partners. Note that partners have to support managing the choreography instance identification in SOAP based message communication (Section 5).

4. MODELING OF MONITORING AGREEMENTS

The monitoring agreement is an XML document consisting of two types of definitions: resource event definitions and complex event definitions. Resource event definitions are specified based on the choreography descriptions and complex events are defined based on other events. Complex events can serve as basis for specification of SLAs and BAM solutions. One could for example specify service level parameters and service level objectives based on complex events which contain corresponding metric values. That is however out of scope of this paper.

4.1 Definition of Resource Events

Resource events are defined based on the abstract BPEL process models in the BPEL4Chor choreography. A resource event definition specifies the following three elements:

- *Monitored Resource:* Firstly, we have to specify which process resource should be monitored and for which state of the resource the event should be published. The resource is identified by pointing to the corresponding BPEL4Chor elements. Monitored resources we are interested in are the instances of the BPEL process, activity, scope, and variable. The state models (e.g., *started*, *completed*, *terminated*, *compensated*, and corresponding transitions) for these resources are not standardized. We use the state models defined in [5]. The resource identification will result at process runtime in corresponding resource identifiers which are transported in the event and are needed for event correlation, as discussed in Section 5.

- *Process Data:* Optionally, one can specify which process data (defined as BPEL variable) is to be part of the event. The data is read at the moment of event publishing.

- *Target message queue or pub/sub topic:* Finally, one has to specify a message queue or a pub/sub topic to which the event is to be published. If the resource event is to be published to a partner only under a certain filtering condition, e.g., some attribute value contained in the event or some other events, an additional complex event has to be created which is created based on this condition (discussed further below). The access to the queue or topic can be restricted to certain participants by specifying their names; the concrete realization mechanism for access control, needed credentials etc. have to be specified separately.

```
<monitoringAgreement
  xmlns:chor="http://purchaseOrder/choreography"
  xmlns:reseller="http://purchaseOrder/reseller">
 <resorceEventDefinitions>
  <resourceEventDefinition name="OrderReceivedEvent">
   <monitoredResource
     choreography="chor:orderChoreography"
     process="reseller:ResellerProcess"
     scope="process"
     activity="reseller:ReceivePO"
     state="completed"/>
   <data>
    <processVariable name="order"
      variable="purchaseOrder"/>
   </data>
   <publish>
    <queue name="purchaseOrder.reseller"
      access="reseller"/>
   </publish>
  </resourceEventDefinition>
  ...
 <resorceEventDefinitions>
 ...
<monitoringAgreement>
```

Listing 1: Resource Event Definition

Listing 1 shows a *resource event definition* for the `OrderReceived` resource event. It is specified by pointing to the `Receive PO` activity in the reseller process model. The event is to be published when the corresponding activity is `completed`. In addition, the event should contain the data from the `purchaseOrder` variable. It is published to the queue which can only be accessed by the reseller. As there are several customers and shippers as potential participants in this choreography, we cannot simply give access to this queue to all participants. Further below, in Section 4.2, we will define a complex event which sends this event to the customer who actually requested this order.

4.2 Definition of Complex Events

Complex events are specified by correlating and aggregating existing events. Event correlation and aggregation is a well-known topic in the area of complex event processing (CEP) and there are different languages available for the specification of complex events [8]. In our case, we have decided to use the language of ESPER[1], which is the CEP implementation we have used in our prototype (Section 5.2). But alternatively any other language could be used instead, the choice being dependent on aspects such as language expressivity needed. Note that we use the term *complex event* for an event which results from using a CEP statement over one or more events; we do not further distinguish between more fine-grained meanings of complex, composite, and derived events as in some other works.

```
<monitoringAgreement
  xmlns:chor="http://purchaseOrder/choreography"
  xmlns:reseller="http://purchaseOrder/reseller">
 ...
 <complexEventDefinitions>
  <complexEventDefinition providedBy="reseller"
    name="CustomerAOrderReceivedEvent"
    choreography="chor:orderChoreography">
   <consume>
    <queue name="purchaseOrder.reseller"/>
   </consume>
   <eventAggregation resultType="FILTER">
    <statement><![CDATA[
     SELECT a
     FROM PATTERN [
      EVERY a=ResourceEvent(name="OrderReceivedEvent"
      AND variables('order').customer="customerA")]
    ]]></statement>
   </eventAggregation>
   <publish>
    <queue name="orderChoreography.customerA"
      access="customerA"/>
   </publish>
  </complexEventDefinition>
  ...
 </complexEventDefinitions>
<monitoringAgreement>
```

Listing 2: Complex Event for Event Filtering

The complex event definition consists of an event aggregation statement and the target topic definition. In addition, we have to specify by whom the aggregation is performed and published on the target queue or topic (`providedBy` attribute). The reason is that we have to avoid that several partners perform this aggregation, as this would lead to a duplication of events. The event aggregation statement uses the CEP language to construct a new complex event out of already defined resource events and complex events. Therefore, we first specify from which queues or topics these existing events are `consumed`. Later, when referencing those events (correctly speaking: event streams) in the `eventAggregation` statement, we use the names from the corresponding event definitions. Considering monitored resource identifiers needed for correlation of events (see Section 5.1 for more details) we use the following naming scheme: cid stands for choreography ID and ciid for choreography instance ID, pid for process ID and piid for process instance ID, sid for scope ID and siid for scope instance ID, aid for activity ID and aiid for activity instance ID.

[1]http://esper.codehaus.org

Complex events definitions are specified recursively based on resource events to achieve two purposes: (i) event filtering and (ii) event aggregation in order to evaluate complex process metrics. In some cases, event filters have to be defined for resource events in order to ensure that the events are delivered to the right participants. Consider in our example the `OrderReceivedEvent` (Listing 1). For privacy reasons, it should only be visible to the customer which placed the order and not to other potential customers which are also defined as participants in the choreography (but that do not participate in this particular choreography instance). If we assume that the `customerID` is part of the `purchaseOrder` variable then we can define an event filter as shown in Listing 2. Only those `OrderReceivedEvents` which contain the correct customerID are placed into the queue `orderChoreography.customerA` accessible only by customerA.

```
<complexEventDefinition providedBy="reseller"
  name="CustomerAOrderFulfillmentTime"
  choreography="chor:orderChoreography">
 <consume>...</consume>
 <eventAggregation resultType="COMPLEX">
  <statement><![CDATA[
   SELECT
    abs(b.timestamp - a.timestamp) AS metricValue,
    "ms" AS unit,
    a.resource.ciid AS ciid
   FROM PATTERN [ EVERY
    a = ResourceEvent(
      name="CustomerAOrderReceivedEvent")
    -> b = ResourceEvent(
       name="CustomerAShipmentReceivedEvent"
       AND resource.ciid = a.resource.ciid) ]
  ]]><statement>
 </eventAggregation>
 <publish>
  <queue name="orderChoreography.customerA"
    access="customerA"/>
 </publish>
</complexEventDefinition>
```

Listing 3: Complex Event for Metric Computation

Besides event filtering, another important use case for complex events is evaluation of process metrics. In Listing 3 we define a complex event `CustomerAOrderFulfillmentTime` which contains the corresponding metric value in the attribute `metricValue`. In addition it contains the attribute `unit` and the choreography instance identifier. The metric value is calculated by correlating two events already defined, namely `CustomerAOrderReceivedEvent` and `CustomerAShipmentReceivedEvent`. These events are correlated based on choreography instance IDs and then their timestamps are subtracted. The result event is published to the corresponding queue by the reseller who also performs this event aggregation. Note that obviously such a definition results in one result event per choreography instance, i.e. an event stream.

5. MONITORING OF CHOREOGRAPHIES

After the monitoring agreement is created, it is deployed to each partner's infrastructure. The partner thereby extracts from the agreement the events it has to provide and configures its middleware, e.g., the BPEL engine using a deployment descriptor to provide resource events, and the CEP engine to provide complex events. It also subscribes to topics or queues where he receives events from other partners. A possible realization is described in Section 5.2.

5.1 Event Correlation in Choreographies

In order to be able to perform event correlation for monitored resources, corresponding resource identifiers have to be included in events. Consider, for example, the calculation of the order processing duration between the activity `Receive PO` and `Receive Shipment Acknowledgment` in our scenario. For each of those activities an event stream is created. Obviously, we need to correlate events belonging to the same purchase order, i.e. the same choreography instance. Thus, events have to contain identifiers of the corresponding monitored resource. In this case, each event belongs to a certain activity instance (note that in general there can be several activity instances per activity if that activity is contained in a loop). An activity instance belongs again to a process instance. However, in this case those two identifiers (piid and aiid) are not enough as those two activities are part of different process models. In our case, in order to be able to correlate the corresponding two events, we have to correlate on the choreography instance level. Thus, the events have to contain an identifier which identifies the choreography instance.

For identifying monitored resources, either technical or business IDs are needed. In the case of process models which are realized as executable BPEL processes, for example, the BPEL engine assigns technical process instance IDs to process instances, scopes and activities (however not to choreographies). For a choreography instance, in the general case also an identifier is needed. In some special cases, correlation could be done based on business identifiers transported in BPEL messages. For example, if the orderID is known to the shipper, then it can be sent in the `Order Shipped` event and thus correlated with the `Order Received` event. However, in the general case this cannot be ensured. In synchronous invocations (BPEL invoke with input and output) the correlation between the sent and replied message is done on protocol level, e.g., SOAP/HTTP and not based on message payload (which does not necessarily contain needed identifiers). Assume for example, the synchronous invocation of the reseller process to a warehouse process to check whether all products are in stock. In that case the orderId is not necessarily part of that message. If now the reseller subscribes to the events of the warehouse those events have to contain a technical identifier.

```
<soap:header>
 <chor:choreography
   xmlns:chor="http://iaas/monitoring/choreography">
  <chor:cid>
    {http://.../choreography}orderChoreography
  </chor:cid>
  <chor:ciid>
    {...}orderChoreography/2009-11-02-12:05:21:005
  </chor:ciid>
 </chor:choreography>
 ...
</soap:header>
```

Listing 4: Choreography-ID in SOAP Header

This is why we need in the general case a technical identifier of the choreography instance which is transported on protocol level. For SOAP-based communication this identifier can be transported in the SOAP header. Obviously, the corresponding middleware, e.g., BPEL engine and service bus, has to be adapted to include and read the identifier during message exchanges. It also has to be transported in events. Listing 4 shows an example SOAP header with the choreography instance identifier. It is created when the first process instance of the choreography instance is created, in our case when `Send PO` in the customer process has started. It contains the choreography ID + timestamp of creation. Note that this approach also works in case of multiple *alternative* start activities; it does however not support multiple start activities triggered in parallel at different participants for the same choreography instance. The semantics of the latter case is not described in the BPEL4Chor specification and we have not found any use case for it.

5.2 Implementation

We have implemented the approach as shown in Figure 3. It shows the implementation from the point of view of one partner in the choreography. The prototype implementation is based on the Apache ODE BPEL engine[2] and the ESPER event processing framework.

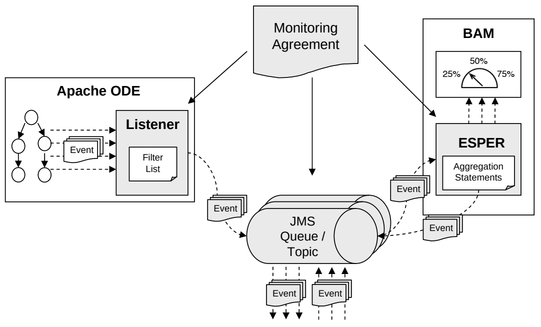

Figure 3: Tool Support

The monitoring agreement XML document which we create by hand is deployed to both an event listener in the ODE engine and the ESPER engine which is part of a BAM tool with a GUI for displaying metrics. The resource event generation is performed by an event listener which we have implemented as part of the ODE engine. It is configured by reading the monitoring agreement document and extracting the resource event definitions relevant for this participant. As the process instances are executed, it receives all internal ODE events resulting from the process execution and filters them according to the resource event definitions from the monitoring agreement. Information from internal ODE process events is taken and augmented with needed process data and choreography instance identifier read from the process instance context and the corresponding (external) resource event is created. It is sent to a JMS queue or topic as defined in the monitoring agreement. We have used Apache ActiveMQ as our JMS implementation[3]. The complex event generation and receipt of events from other participants is implemented by the BAM tool in our architecture. It uses ESPER as the underlying CEP framework and contains a GUI for displaying received events and has some dashboards for showing calculated metrics. Event aggregation statements from the monitoring agreements are registered as

[2]http://ode.apache.org
[3]http://activemq.apache.org

ESPER statements and produce complex events which are published on specified queues or topics. Finally, the last part we had to implement is dealing with the choreography instance ID (ciid). Therefore, the ciid is read from and written into SOAP headers in message interactions with other choreography participants. It is saved in the process instance context and is also propagated to the event listener which then writes the ciid into events. A new ciid is created if there is no ciid in the received message which is the case during instantiation of the first process instance of the choreography.

6. RELATED WORK

As already explained in the introduction, state of the art event-based process monitoring solutions are based on BAM technology and focus on intra-organizational processes. There exist several research approaches [1, 2] and products [11] which deal with evaluation of process metrics in near real time and their presentation in dashboards. They all have in common that events are emitted as the process is executed, collected by a process monitor and evaluated in near real time. Some solutions focus on monitoring of BPEL processes [1, 2], while others are more general and support an extensible architecture via event adapters [11]. These approaches are similar to ours in that they also use an event-based approach based on BPEL processes. However, they focus on single BPEL orchestrations and do not deal with monitoring of choreographies in a cross-organizational setting. The only approach we are aware of which deals with monitoring of BPEL processes in a cross-organizational setting is presented in [7]. Thereby, a common audit format is presented which allows processing and correlating events across different BPEL engines. In our approach, we also assume that the participants have agreed on state models of BPEL resources and resulting resource event definitions. In addition, we deal with complex events specification and event correlation in choreographies using a choreography instance identifier which is not supported in [7].

Service Level Agreements (SLA) are similar to our problem in that they involve monitoring in a cross-organizational setting. Thereby mostly two partners, the service consumer and the service provider, agree on certain service QoS, typically technical characteristics such as availability and response time [6]. The commonalities with monitoring in our context are that in an SLA partners also agree on metrics and how they are to be monitored. However, in our case the focus is on event-based monitoring of process metrics across participants in a choreography which is not being dealt with in frameworks such as WSLA focusing on QoS measurements. Our approach, however, could be extended towards specification of SLAs based on the monitoring agreement.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an approach to event-based process monitoring based on service choreographies. A monitoring agreement is created which defines events each partner in the choreography has to provide. We have distinguished between resource events which are defined based on a BPEL4Chor choreography description and complex events using a CEP language. We have also shown the need for a choreography instance identifier for event correlation and how it can be included in a SOAP based communication.

Throughout the paper we have already discussed several possible extensions for our future work. In this paper we focused on an event-based monitoring approach (a.k.a. push model). We will extend this approach by enabling also partners requesting monitoring information on demand (a.k.a. pull model). Furthermore, we will explore dealing with a dynamic set of unknown participants at design time. At the moment, in order to ensure privacy, static queues and topics with corresponding access rights are defined in the agreement; we plan to extend this towards creating dynamic queues and topics for participants which are not known before runtime. Finally, we will deal with a bottom-up approach to specification of monitoring agreements as discussed in Section 3 and we will explore the usage of Web service Distributed Management (WSDM) set of specifications as underlying monitoring infrastructure.

Acknowledgments The research leading to these results has received funding from the European Community's 7th Framework Programme under the Network of Excellence S-Cube - Grant Agreement no. 215483.

8. REFERENCES

[1] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *Proceedings of the IEEE International Conference on Web Services(ICWS'06)*, pages 63–71, 2006.

[2] L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *Proceedings of the 3rd International Conference of Service-Oriented Computing (ICSOC'05)*, pages 269–282. Springer, 2005.

[3] G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *ICWS*, Salt Lake City, USA, July 2007.

[4] G. Decker, O. Kopp, F. Leymann, and M. Weske. Interacting Services: From Specification to Execution. *Data & Knowledge Engineering*, 68(10):946 – 972, 2009.

[5] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, and F. Leymann. BPEL Event Model. Technical Report 2006/10, University of Stuttgart, Germany, November 2006.

[6] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Netw. Syst. Manage.*, 11(1):57–81, 2003.

[7] S. Kikuchi, H. Shimamura, and Y. Kanna. Monitoring Method of Cross-Sites' Processes Executed by Multiple WS-BPEL Processors. In *CEC/EEE*, pages 55–64, 2007.

[8] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Professional, May 2002.

[9] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 11, 2007.

[10] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.

[11] U. Wahli, V. Avula, H. Macleod, M. Saeed, and A. Vinther. *Business Process Management: Modeling Through Monitoring Using WebSphere V6.0.2 Products.* IBM, International Technical Support Organization, 2007.

[12] M. Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.