Towards a Methodology for the Engineering of Event-driven Process Applications [⋆]

Anne Baumgraß[1], Mirela Botezatu[2], Claudio Di Ciccio[3], Remco Dijkman[4], Paul Grefen[4], Marcin Hewelt[1], Jan Mendling[3], Andreas Meyer[1], Shaya Pourmirza[4], Hagen Völzer[2]

[1] *Hasso-Plattner-Institut, University of Potsdam, Germany*
[2] *IBM Research – Zurich, Switzerland*
[3] *Institute for Information Business at WU Vienna, Austria*
[4] *Eindhoven University of Technology, The Netherlands*

Abstract. Successful applications of the Internet of Things such as smart cities, smart logistics, and predictive maintenance, build on observing and analyzing business-related objects in the real world for business process execution and monitoring. In this context, complex event processing is increasingly used to integrate events from sensors with events stemming from business process management systems. This paper describes a methodology to combine the areas and engineer an event-driven logistics processes application. Thereby, we describe the requirements, use cases and lessons learned to design and implement such an architecture.

Key words: Event-Driven Process Applications, Business Process Management, Architecture Design, Methodology, Logistics

1 Introduction

Traditionally, Business Process Management Systems (BPMSs) execute and monitor business process instances based on events that stem from the process engine itself or from connected client applications. However, recently, successful applications of the Internet of Things such as smart cities, smart logistics, and predictive maintenance emerge that include and provide sensors tracking objects via Global Positioning System (GPS) or Radio-Frequency Identification (RFID), measuring the temperature, the energy consumption, or other types of data. Thus, information from the environment in which processes are executed is available but often not considered in the design of traditional BPMSs [1].

Such external applications offer new possibilities to control and evaluate the business process execution, yet they require a novel concept of integration with a BPMS. Complex Event Processing (CEP) is often considered as a suitable technique for tackling this challenge [2], especially in logistics [3].

[⋆] The research leading to these results is part of the GET Service project and has received funding from the European Commission under the 7th Framework Programme (FP7) for Research and Technological Development under grant agreement 2012-318275.

In the GET Service project[2], we analysed typical logistics scenarios, explored their environments, and evaluated BPMSs for their execution and monitoring. GET Service aims at techniques and systems to plan transportation routes more efficiently and to respond quickly to unexpected events such as adverse weather or strikes, during transportation. In recent studies [4], disruptions due to such events mainly cause a loss of productivity and revenue as well as an increased cost of working. Therefore, timely notifications are important in logistics to reduce the impact. Nowadays, these disruptions are handled manually and often by phone – an ineffective and not reliable (in terms of fast reaction and prevention) source of interaction. In this paper, we document how the aim of the project to automate these notifications is addressed, specifically by the selection of a process engine in combination with a CEP system and a process modelling tool that got extended to support the required functionalities like automatic event query generation.

In particular, we present – in terms of a methodology – the course of actions we conducted from the description of the logistics scenarios to the iterative design and implementation of the systems supporting these scenarios. In this line, we present an architecture that supports the execution and monitoring of the business processes across distributed BPMSs and the consideration of heterogeneous event sources. It is mainly based on an analysis of transportation scenarios from practice. Although stemming from logistics, the architecture can be generalized and also used in further domains. Its core components make use of novel concepts for annotating process models with which the event subscription can be automated. As a proof-of-concept, we implemented the architecture such that progress and violations during process execution can be effectively monitored for process-internal and environmental events; the implementation may be found in [5].

Against this background, the remainder of this paper is structured as follows: Section 2 briefly introduces the methodology used before Section 3 discusses the use case for this paper. Section 4 presents an architectural overview of our process-based real-time execution and monitoring approach which is mainly based on use cases similar to the one presented in Section 3. Thereby, Sections 4.1, 4.2, and 4.3 zoom in to three important components of our architecture: process modeller, process engine, and event processing engine. Finally, Section 5 presents the conclusion and outlook.

2 Methodology

The proposed methodology results from the GET Service project that aims to improve transport planning and execution in terms of transport costs, CO_2-emission, and customer service. Thus, it requires the design of a service platform for joint transport planning and execution. As starting point, a basic understanding of the logistics domain and its scenarios were established. In total, five usage scenarios were identified, each one depicting a particular logistics chain from different industries and focusing on different transport modes and stakeholders [6]. For each scenario, several use cases, influences of events, and success criteria were defined. These served as input to specifically define

[2] `http://www.getservice-project.eu`

processes including its data model, roles, and actors. For further investigation in the project, we specifically decided to support three use cases [7]: 1) Export of containers through a port requiring an efficient synchronization of transport participants, 2) Airfreight transportation including the possibility of shifts from one airport to another, and 3) Synchromodal transport synchronizing multiple transports for the same order at once. In the remainder of the paper, we consider the process of a freight-shift in use case (2) to reflect our requirements, the corresponding architecture, and its implementation. It is presented in Section 3.

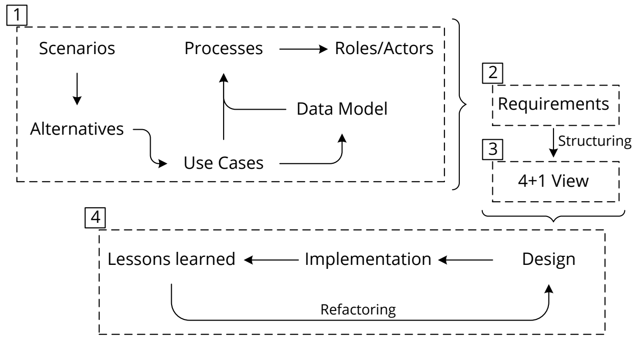

Figure 1. Procedure of stepwise refinement of information towards the final artefact: the implemented software system

Figure 1 shows the procedure of our methodology. The main input for the requirements analysis (step 2) were the process models we created together with domain experts and users from this field in step 1. Using those we were able to exactly identify the right roles and actors as well as a joint data model for all use cases. In the requirement analysis, we determined the functional and non-functional aspects of the GET Service platform necessary for transportation planning. Specifically, we defined how planning relies on historic and real-time information and in which way this information should serve as its input. For example, how real-time information like an accident of a truck is recognized, correlated with a transport plan and changing it as well as how this information is forwarded to the user.

After eliciting the requirements, they need to be transitioned into a system architecture. Thereby, views on processes, system interfaces, domain model, system functionality, and the utilized physical hardware must be considered – a wide spectrum of information.

We used Kruchten's 4+1 view model [8] to structure all requirements in step 3 as input to design, implement, and refactor GET Service's architecture in step 4. This model combines aspects of architectural models, layers, and principles from the field of software and systems engineering. As such, it is used to create distinct views for each of the stakeholders' viewpoints. Figure 2 shows the application of this model to our project and the above mentioned views.

Considering the example of the functional requirement to provision real-time information, the description of such is dealt with in the logical view. From those, the process models are constructed in the process view to deal while the scenarios are used to derive the specific behaviour. The components enabling the functionality are designed in the

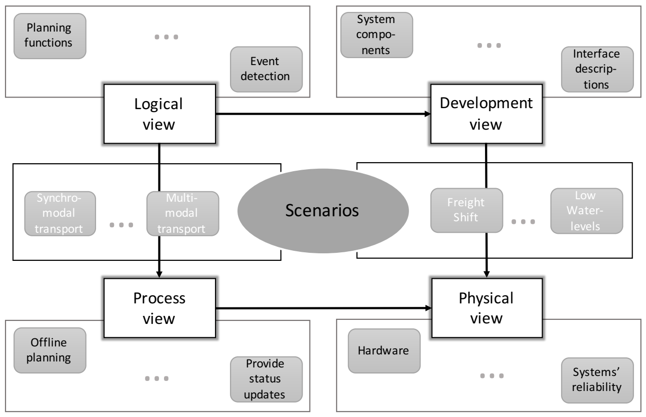

Figure 2. Kruchten's 4+1 [8] model applied to GET Service

development view. Finally, the physical view shows the concrete implementation and the set-up of the infrastructure to enable the provisioning of real-time information. To meet the interests of this workshop, we focus, in the remainder of this paper, on the parts of GET Service that deal with execution and monitoring of logistics processes.

3 Running Example of a Freight-Shift in Transportation

As a example, we consider the part of a logistic process instance pertaining to the transportation of a container from a storage area in Nice to a distribution centre located in Frankfurt. The whole multi-modal transport plan comprises a first truck-based leg from the French storage area to Nice Côte d'Azur Airport, then a second aircraft-based leg brings the cargo to Frankfurt Airport, and finally a truck-based leg towards the distribution centre in Germany. Such a transport is managed by a logistics service provider (LSP) that can utilise a fleet of trucks of its own, distributed Europe-wide, or some subcontractor, another LSP. Airfreight, however, is usually handled from a commercial cargo airline company which thus is in charge of the air-based leg of the transport.

From recent studies [4] and the scenarios we surveyed in the GET Service project we learned that numerous disruptions might occur even in such simple transports as described here. Not all of them are known a-priori but happen unexpectedly. For example, an air-plane may have to unexpectedly land at a different airport due to adverse weather or a strike. Alternatively, an air-plane might suddenly have more transportation capacity available, because of transportation order cancellations. In the end, both occurrences lead to a shift of transportation capacity or demand from one location to another. However, nowadays, real-time information of transport progress or disruptions are often communicated late, incomplete, or by no means at all among the affected actors.

In our process, for instance, a strike at Frankfurt airport could force the pilot to divert the flight to another location, namely Brussels Airport. This would certainly compromise the successful completion of the multi-modal transportation activity, because empty trucks would wait for the air-plane to land in Frankfurt, since the logistics service provider did not get informed about the diversion. This would not happen before the actual landing in Brussels, with clear negative effects on *(i)* productivity, due to the longer waiting times and empty trucks, *(ii)* costs, caused by the re-routing or re-booking of additional trucks to move the containers from Brussels to Frankfurt, and *(iii)* timeliness of the delivery, due to the fact that compensation actions would be taken after landing.

Obviously, freight-shifts and other events influencing the completion of the process should be detected as quickly as possible to properly act on them. This means, they have to be considered by the process engine enacting the process model. In consequence, the possible occurrence of events and a reaction to them have to be integrated into the process model. With current technology, this would restrict the monitoring only to events known at design time, require the determination of a reaction, which might differ from instance to instance, and increase the complexity of process models drastically. Furthermore, the process engine must collect events relevant for the process during execution of a process and evaluate the impact of them. In contrast to the freight-shift, for example, a small congestion on a road might only cause a delay for the truck that does not influence the transport chain, while a problem with the motor of the truck might enforce the planner to pick another truck for the transport. Section 4 details the architecture and highlights the requirements leading to its design and the corresponding implementation. The system including the interaction between different components is provided as screencast at `http://youtu.be/JE2Df7iaERk`.

4 Design of an Architecture for Event-driven Logistics Processes

The Workflow Management System (WfMS) Reference architecture [9] served as blueprint for the design of our event-driven logistics process architecture. As central component it shows the workflow enactment service that is connected via 5 interfaces to other components for modelling (IF1), for controlling client applications and devices (IF2), for invoking external applications (IF3), for monitoring and administration (IF4), and also for invoking other WfMSs (IF5). Currently, WfMSs are designed to operate with static control structures. Dynamic environments, such as logistics with a lot of events (e.g. weather, congestions, or strikes influencing the process execution), require more flexibility and interfaces to consider external events. Therefore, dynamism is the key requirement in our architecture.

Based on investigations on the connection between events and process models in [3], we added a new interface from the workflow enactment service to the event engine fostering flexibility. In this setup, for each task in a process model, not only the workflow execution semantics need to be evaluated, but also the associated event notifications have to be considered. Therefore, this design alternative may reduce the performance of the whole system. However, it can increase the dynamism of our architecture. This design allows all the tasks in a process model to communicate with the event processing engine, and consequently, one task can be executed 'both' manually by a user in the user interface of the process engine or automatically based on high-level events.

From this high-level we designed the architecture for connecting environment sources and consider them during process execution as shown in Figure 3. This architecture also shows the necessary components to enable event-driven logistics processes and their interfaces: *(i)* The *Event Engine*, responsible for event processing, definition of high-level events, and provisioning of subscriptions mechanisms, *(ii)* the *BPMS*, which allows to model (*Modeller*) and enact (*Process Engine*) processes, and *(iii)* the set of *Event Sources* providing the information processed by the event engine.

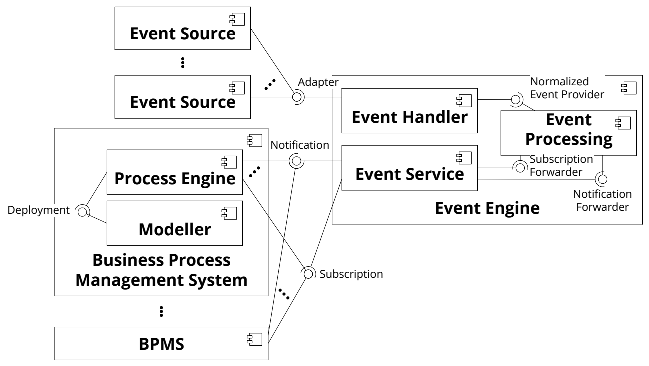

Figure 3. High-level architecture for real-time monitoring of business processes through CEP

This architecture shows how real-time event data can be correlated automatically to real-time process data and vice versa through publish subscribe mechanisms. Several event sources provide event streams, e.g. GPS locations, which input event data into the *Event Engine*. This may happen constantly or in chunks. The *BPMS* allows to model business processes which are then enacted within a *Process Engine*. The event information is automatically correlated to the actual process instances – the run-time equivalents of process models – such that subscriptions must be defined that are provided to the *Event Service* which in turn provides notifications to the affected process instances for each set of events corresponding to a subscription.

In the following sections, we discuss the requirements, the design, the implementation, and the lessons learned for the three main components in our architecture: the modeller, the process engine, and the event engine. This is in line with the steps displayed in the lower part of Figure 1 and the logical, development, and physical views in Figure 2.

4.1 Process Modelling

In general, a process model can be used to integrate different IT systems and services based on Service-Oriented Architecture (SOA) and a process execution engine.

Requirements Based on the use case described in Section 3, the process modelling language must *(1)* support modelling of tasks, events and messages occurring in a supply chain, *(2)* have execution semantics and suitable engines supporting this semantics, *(3)* introduce roles for different parts in the process model to specify the responsible client or external IT system for its execution, *(4)* include a specification that the progress of some tasks is determined by external information, *(5)* provide means to specify time- and location-related constraints in the process model that are relevant for the process engine to reveal the adherence of process execution with the process model, and

(6) support semantics for a process engine to use process models not only for execution but also for status tracking and visualization of disruptions.

In our literature review, we observed that there is no *off-the-shelf* process modelling language that fully supports all requirements [10]. Thus, we decided to extend the industry standard Business Process Model and Notation (BPMN) [11], since it is widely supported by tools and engines, has a suitable expressiveness, provides standard mechanisms for its extension, and has high emphasis on both, the design and analysis as well as the process automation of use cases. BPMN provides a XSD-specification of its metamodel. Thus, BPMN-conform process models can be exchanged via XML.

Design To adequately address the requirements of logistics processes, such as the detection of a flight diversion from our use case, we enriched the BPMN metamodel with annotations that capture the time and location constraints of a logistics process as well as annotations to consider the internal and external event sources either driving the process or influencing its execution. Details on the extension are provided in [10].

The annotations may be explained using the automatic generation of event subscriptions from process models. For the purpose of automation, we pre-define various event subscription templates that are then filled by the modeller component with process model information (e.g. the deadline to execute a task) and extended by the process engine with process instance information (e.g. the truck that actually transports the cargo). Note that these templates are written in Esper [12] to be conform with the event processing language of the event engine. An example of such a subscription template is shown in Listing 1. It aims at notifying about strikes (`type="strike"`) at a specific destination airport before a certain deadline is reached. Case-dependent information is marked by preceding '$'. Thus, while the name of the destination airport (`$destination`) and the deadline (`$deadline`) in this example should be extracted from the process model, the specific case for which this query is relevant (`$caseId`) must be set by the process engine itself. Given the specific task "Drive to destination" in Figure 4 the process engine uses the annotations in the process model and subscribes to be notified about strikes at 'Frankfurt' before '2015-07-04T08:00' (resp. latest arrival of the truck in Frankfurt).

Listing 1. Event subscription template for strikes at an airport.

```
SELECT type,timestamp,description,url,title,latitude,longitude, $caseId as
    caseId FROM Warning where type="strike" and description like "%$
    destination%" and timestamp.before($deadline)
```

An event subscription is associated with a BPMN element, encapsulated in the documentation. The types of events that result from an event subscription are determined by the event type given in the documentation ((`<eventType>Warning</eventType>`)). It furthermore includes a scope defining when the query has to be activated and how the responses of the event engine to this query have to be interpreted, see Listing 2. The scope in this example does not trigger any status update of the task (`trigger="false"`) and the observation of corresponding events is within the task execution (`<startTask>Task2</startTask>`

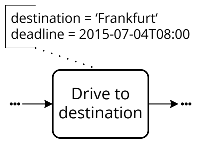

Figure 4. Annotated task "Drive to destination".

`<endTask>Task2</endTask>`), i.e. the subscription starts when the truck starts driving and ends when the truck arrives at the airport, which can be captured in further queries. Obviously, in this example, the modeller, the process engine and the event engine have to have a common understanding of the annotations and how to understand their content.

Listing 2. Excerpt of an event subscription in the XML-based T-BPMN process model.

```
<task id="Task2" name="Drive to destination">
 <documentation> <queryAnnotation> <query>
  <queryText> [Query of Listing 1] </queryText>
  <eventType>Warning</eventType>
  <scope trigger="false">
   <startTask>Task2</startTask>
   <endTask>Task2</endTask>
  </scope>
</query> </queryAnnotation> </documentation> ... </task>
```

Lessons Learned The provisioning of subscription templates is essential in our approach, since the low level of modelling competence of modellers in process documentation projects is a well-known problem in the Business Process Management (BPM) field [13]. From the practical experience in the GET Service project, further challenges arise when users are not used to model process models at all. Furthermore, closely related to our event engine (see Section 4.3), we defined aggregation rules that hide the complexity of event processing from the user to ease the utilization and understanding of event subscriptions. For instance, we specify rules in the event engine that produce high-level events whenever a truck reaches a destination. Thus, the modeller does not need to define queries comparing all truck positions to reason about destination arrival himself. Instead, she can simply subscribe via “Select * from ArrivedAtTransportNode(operatorId=$truckId)”.

4.2 Executing Annotated Process Models

In this section, we review the features that need to be supported by our process engine.

Requirements We employed a systematic review approach to create a list of all possible process engines that can execute any kind of process model. We considered both service orchestration engines or business process management execution engines. Based on the general requirements derived from the components functionality, the architecture, and the required interfaces, we identified 6 generic criteria the 36 identified engines are evaluated against: *(i)* Does the system have an open-source license? *(ii)* Does the system have an in-depth documentation? *(iii)* Does the system support BPMN language? *(iv)* Has the system been implemented in Java? *(v)* Does the system have an active developer community for communication? *(vi)* Does the system support runtime adaptability?

The third criterion considers our decision for BPMN as business process modelling language as discussed in Section 4.1. The sixth criterion evaluates the ability to change the process model during runtime of the system. This is required due to re-planning activities in logistics. In [14], we discussed the process engine evaluation in detail.

Summarized, based on our requirements, we chose the Activiti [3] process engine as our base process engine.

Design The *process engine* requires algorithms to parse and interpret the annotated process models (cf. Section 4.1) and needs to ensure that activities specified therein are properly executed and monitored. To achieve this, the process engine enacts process models, by generating *process instances*. A process instance captures for each activity *(i) when*, *(ii)* by *whom*, and *(iii) how* its state can be updated.

Most interesting for the interaction between the introduced systems, all notifications the process engine receives are extracted from the annotated process model and extended with process instance information. While notifications about progress or status updates do not affect the design of the process model, the process engine may receive notifications about violations in the business process, as either actual or predicted ones. Please note that violations often involve updates both in the design and execution of the process model. Thus, we require the implementation of a dynamic reconfiguration of process instances which we discuss in detail in [15].

Implementation We extended the Activiti [16] process engine in three main aspects: (i) We allow full support of runtime adaptations of the process; (ii) we integrate the process engine with the event processing engine in order to react to the external events; and (iii) we provide additional access to the process engine through external devices, e.g. mobile clients. As mentioned in the previous section, an algorithm has been developed in order to apply the instance migration after process adaptation and address extension (i).

Extension (ii) was discussed in Section 4, where we discussed the integration of a WfMS with an event processing engine. We designed an additional interface to facilitate this extension. As consequence of this integration, the Activiti engine can subscribe to listen to predefined *query notifications* (e.g. position update of a vehicle) it can produce events about the executions (e.g. status of each task). Therefore, the process engine can always publish events; however in order to listen to a specific event, the prerequisite is the existence of a process model that is sufficiently annotated with the query notifications (cf. [10]).

Extension (iii) to the Activiti engine provides different RESTful services that contain both: information about the execution of a process instance and query notifications. Therefore, any web-based client (e.g. mobile device) can employ these services in order to interact with the WfMS remotely.

Lessons learned We revealed that the assignment of tasks to users may differ from the user who is interested in the notifications for a task; e.g. a driver executes some driving while the planner monitors this execution. For the former, we utilize the concept of swimlanes (provided by BPMN and Activiti) while for the latter, we introduced additional policies by extending the developed interface IF6 (cf. Figure 3) such that it now supports both: user specific event queries and an integration of our WfMS with our event processing engine (extension (iii)).

[3] `http://www.activiti.org/`

4.3 Event Processing

Event sources are meant to provide streams of data, i.e. a series of updated digital information objects (henceforth, *events*). In our context, events pertain to the applicants, devices, and means of transportation under control that are utilized for executing a logistics process. Knowledge can be extracted from events that *(i)* permits to monitor the carry-out of related activities, *(ii)* detects anomalies in their execution, and *(iii)* predicts possible disruptions in the process execution.

Requirements Corresponding to [17] and the use case and requirements described before, an event engine must provide: *(1)* a generic interface to different sources which streams events concerning trucks, air-planes and their environment, *(2)* technologies to normalize data in heterogeneous formats for processing, *(3)* support to define high-level events, *(4)* a publish-subscribe interface for other systems to receive events or respective notifications about events, and *(5)* perform ex-ante predictions on the future evolution of the process instance.

Design The *Event Handler* is designed to read multiple sources by means of dedicated *Adapters* to publish them as uniform events for event processing. This entails that the event engine not only has to provide the possibility to register queries to be informed on events of interest, but it also needs to be capable of automatically correlating events to transport processes. In our architecture, event processing is based on the subscriptions forwarded by the process engine that derived them from sufficiently annotated process models, e.g. for progress updates and violation detection. For a correlation with external event data, instance-specific information is required in the subscriptions, e.g. the container id or the destination of a transport.

To match high-level events to subscriptions of the process engine, the event engine must also be capable of aggregating events of finer granularity, such as the subsequent positional updates of the air-planes, to more informative ones, such as the changes of speed, altitude, and direction in a time interval. To this end, techniques coming from the area of Complex Event Processing [18] are exploited.

Finally, the event engine is demanded to predict diversions ahead of time. While predicting violations based on deterministic events such as congestions can be done through subscriptions as shown above, probabilistic predictions correspond to exceptional situations that turn out to be hard to encode by means of human-specified subscriptions. Therefore, we resort on automated classification and regression techniques, well established in the research field of Machine Learning [19]. The objective is to make the event processing module capable of automatically classifying anomalies in harmful or not significant for the correct process execution based on conditions learned through analysis of past executions. When possible, the quantification of foreseen effects on the process are also expressed in terms of, e.g. of delays.

Implementation Technically, the core functionality of event processing is provided by the open source platform Esper[4], which we extended based on the requirements

[4] `http://www.espertech.com/`

identified. We included an event hierarchy and aggregation rules for the logistics domain. Additionally, we included custom function calls for geographic calculations and to access static data on transport nodes. For the management of event sources, aggregation rules, subscriptions, and logistics process specific services, we built a framework around Esper [12]. The event engine, in which Esper is embedded and used in the GET Service project, is called UNICORN[5]. UNICORN is implemented in Java and devises several ways to receive events: through a web service, using the web-based UI, through ActiveMQ[6], and through configurable adapters that pull events from different sources. Consumers, like the process engine in our scenario, subscribe to events by sending Esper queries via a web service. Another web service allows to register transports by passing their parameters, in which case the event engine handles all subscriptions. Notifications are distributed via ActiveMQ queues. Alternatively, the web UI of the event engine can be used to display notifications.

Consider the air-plane from our use case in Section 3. The processing and analysis of events tracing its position would contribute to observe its anomalous behaviour by changing the route and to foresee the disruption of the transportation process [20]. The event monitoring, the anomaly detection, and the disruption prediction are the functionalities that are also encapsulated in the event service component (see Figure 3).

Lessons Learned Rather than be unified under a single general-purpose component to derive or predict high-level events, the dedicated modules can be plugged in. Each module adopts its own algorithm to interpret the current status and recent history of the process instance according to the scope of its predictive analysis. For instance, the module that raises alerts in case of delays of trucks differs from the one utilised for the flight route anomalies. At the same time, they also act as event sources that publish their outcomes as fine-granular events that are then aggregated and retransmitted as prediction events. In the scenario, a diversion prediction would be notified when a given number of consecutive flight route anomalies were received and aggregated by the Event Processing component. This actually reflects the technique adopted in [21], a technique applying event processing in the context of flight diversions within logistics transports.

5 Conclusion

Guided execution and monitoring the status of business processes is of vital importance, if an organization needs to promptly react to occurring problems. We presented a methodology to engineer a logistics process application that is driven by events for execution and monitoring. This allows utilization of information from several event sources, pre-processed and refined by an event engine, to not only execute or monitor processes, but also predict potential problems. We argue that the event subscriptions, necessary for such purposes, must be contained as annotations in the process model which are automatically registered by a process engine with the event engine. Exemplified, we discussed the design of such a system, presented the corresponding implementation, and

[5] `http://bpt.hpi.uni-potsdam.de/UNICORN`
[6] `http://activemq.apache.org/`

highlighted lessons learned for the modelling, execution, and monitoring. The complete implementation and the interaction between the components is provided as screencast provided at `http://youtu.be/JE2Df7iaERk`.

In future work, we aim to establish a reference model for the whole process and describe it along the whole architecture of the GET Service project. We also expect our approach to be generalizable to other domains such as manufacturing or healthcare, in which external events play a central role in the proceeding of business processes.

References

1. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2013)
2. Luckham, D.C.: Event processing for business: organizing the real-time enterprise. John Wiley & Sons (2011)
3. Cabanillas, C., Baumgrass, A., Mendling, J., Rogetzer, P., Bellovoda, B.: Towards the Enhancement of Business Process Monitoring for Complex Logistics Chains. In: BPM Workshops, LNBIP 171, Springer (2013)
4. Business Continuity Institute: SUPPLY CHAIN RESILIENCE 2014 - An international survey to consider the origin, causes & consequences of supply chain disruption (2014)
5. Baumgrass, A., Di Ciccio, C., Dijkman, R., Hewelt, M., Mendling, J., Meyer, A., Pourmirza, S., Weske, M., Wong, T.: GET Controller and UNICORN: Event-driven Process Execution and Monitoring in Logistics. In: BPM Demo track, CEUR Workshop Proceedings (2015)
6. Treitl et al.: Use Cases, Success Criteria and Usage Scenarios, Deliverable report 1.1, GET Service (2014)
7. Baumgrass, A., Dijkman, R., Grefen, P., Pourmirza, S., Voelzer, H., Weske, M.: A Software Architecture for Transportation Planning and Monitoring in a Collaborative Network. In: 16th IFIP Working Conference on Virtual Enterprises. IFIP AICT Series, Springer (2015)
8. Kruchten, P.: Architectural Blueprints - The ”4+ 1” View Model of Software Architecture. Tutorial Proceedings of Tri-Ada **95** (1995) 540–555
9. Hollingsworth, D., Hampshire, U.: Workflow management coalition the workflow reference model. Workflow Management Coalition **68** (1993)
10. Botezatu, M., Völzer, H.: Language and Meta-Model for Transport Processes and Snippets, Deliverable D4.1, GET Service (2014)
11. Object Management Group: Business process model and notation (BPMN) v2.0 (2011)
12. Bernhardt, T., Vasseur, A.: Esper: Event Stream Processing and Correlation (2007)
13. Mendling, J., Reijers, H., van der Aalst, W.: Seven process modeling guidelines (7PMG). Information and Software Technology **52**(2) (2010) 127–136
14. Pourmirza, S., Dijkman, R.: A Survey of Orchestration Engines,, Deliverable report 7.1 , GET Service (2014)
15. Pourmirza, S., Dijkman, R., Grefen, P.: Switching parties in a collaboration at run-time. In: EDOC, IEEE (2014) 136–141
16. Rademakers, T.: Activiti in Action : Executable business processes in BPMN 2.0. First edn. Manning Publications, Shelter Island, NY (2012)
17. Baumgrass, A., Breske, R., Cabanillas, C., Ciccio, C.D., Eid-Sabbagh, R., Hewelt, M., Meyer, A., Rogge-Solti, A.: Conceptual architecture specification of an information aggregation engine, Deliverable report 6.2 , GET Service (2014)
18. Mühl, G., Fiege, L., Pietzuch, P.: Distributed Event-Based Systems. Springer (2006)

19. Mitchell, T.M.: Machine Learning. 1 edn. McGraw Hill series in computer science. McGraw-Hill, Inc., New York, NY, USA (1997)
20. Baumgrass, A., Hewelt, M., Meyer, A., Raptopoulos, A., Selke, J., Wong, T.: Prototypical Implementation of the Information Aggregation Engine, Deliverable D6.3, GET Service (September 2014)
21. Cabanillas, C., Di Ciccio, C., Mendling, J., Baumgrass, A.: Predictive Task Monitoring for Business Processes. In: BPM, Springer (2014) 424–432