Enforcement from the Inside: Improving Quality of Business in Process Management*

Hanna Eberle[2], Stefan Föll[3], Klaus Herrmann[3], Frank Leymann[2], Annapaola Marconi[1], Tobias Unger[2], Hannes Wolf[3]
[1]Fondazione Bruno Kessler, Via alla Cascata 56/c, 38100 Trento, Italy
[2] Institute of Architecture of Application Systems, Universitätsstrasse 38, 70569 Stuttgart, Germany
[3] Institute of Parallel and Distributed Systems, Universitätsstrasse 38, 70569 Stuttgart, Germany
{lastname}@fbk.eu|iaas.uni-stuttgart.de|ipvs.uni-stuttgart.de

Abstract

In this paper we introduce a new modeling tool for constraint handling in the area of workflow technology. The constraint handlers can be used to improve the quality of business processes but without changing already existing business logic. Todays workflow languages provide no possibility to model constraints and the actions in case the constraints get violated explicitly. Fault and event handling mechanisms to react to events not expected in normal executions are only provided by the BPEL language. Using BPEL as workflow language we integrate the constraint handling extension without changing any existing semantics in a smart way. In our approach we use this fault and event handling mechanisms to extend the BPEL language with a constraint handling mechanism. By integrating this constraint handling tool into the BPEL language we provide an approach for quality driven process modeling with the BPEL language.

1 Introduction

Business processes have evolved from rigid structures to workflows highly exposed to the dynamics of the open market and the close interdependencies of business partners. The capability to react immediately on changing conditions in business environments has been recognized as an important factor for the enduring success of business oriented systems. In order to tailor business logic to the actual needs of customers and business cooperations, the flexibility of business processes is considered a cornerstone of today's business management principles. However, the requirements for the development and maintenance of complex business applications have even increased, since business workflows are being increasingly tightly coupled to real world processes. In scenarios like the Smart Factory [15], Logistics Management or Health Care [22], where processes are supposed to react autonomously to context information such as the location of tools or the health of patients, the managed flows of activities are directly situated in real world environments. Workflow management systems are supposed to allow for the continuous synchronization of the execution of business processes with events originating from these environments. For this purpose, the input provided by external observers of environmental information such as context recognition systems directly affects the ongoing process behavior. In this regard, business processes must be aligned to run in compliance to the state of the real world at each point in time in the current context of execution. Therefore, the design of process models must allow for the integration of constraints guaranteeing the validity of business processes based on a clear separation of concerns between business logic and constraint enforcement. These constraints may follow a set of rules necessary for assuring and improving the quality of a process and consequently the resulting outcome of the process (e.g. the manufactured product). As we show in a case study in this paper, these rules are often derived from a set of norms or laws underlying the business processes in real world applications. The detection of constraint violation during run-time and the following adaptation of the business process provide efficient means to perform the necessary steps in order to improve the overall business quality. Based on these insights, we present in this paper a novel approach for the design and execution of business processes being highly adaptive to conditions observed in the real world. For constraint enforcement we regard business processes from the inside allowing constraints to cover multiple related activities of the process model and monitor their validity dur-

*This work is partially funded by the ALLOW project. ALLOW (`http://www.allow-project.eu/`) is part of the EU 7th Framework Programe (contract no. FP7-213339).

ing run-time. Thus, our works extends current approaches, which merely mediate between required and offered quality of service announced by external service providers (e.g. Web Service Level Agreement [16]). In contrast, our approach enables to enforce quality of business throughout process structures by monitoring process-related conditions and treating them as invariants for the enclosing process fragments. Our main contributions are a) the extension of process model design to structure models into sub parts for which compliance rules must be enforced in an individual way b) a way of defining constraints related to external state captured as metrics during run-time as well as the internal execution plan, e.g. the duration of a process instance c) a clear semantics for different choices of handling constraint violation and d) the application of our ideas to BPEL based workflow specification as well as an architecture of an extended workflow engine for the integration of our concepts. As the Web Services Business Process Execution Language (WS-BPEL or BPEL in short) is the de facto standard for composing functionality into a business process, we develop our concepts in strong alignment to the BPEL syntax and semantics, although being applicable to other workflow languages as well.

1.1 Contribution

The aim of this paper is to provide a modeling approach to annotate business process with constraints and an execution model to monitor those constraints during process execution. Additionally the modeler can annotate actions on business processes which have to be performed in case a constraint cannot be hold. To use annotations externalizes the handling of constraints from the core execution of the process logic. As it is shown in Figure 1 first step of the modeling is to model the process itself. Later on the process is annotated with constraints, constraint handling capabilities and metrics as well. Metrics provide the data needed by constraints and the scale basis of the subject to be measured. Metrics can be of various kinds. Simple metrics are e.g. temperature, time but can also be of a more complex kind, e.g. aggregating some context data relevant to real world applications. Constraints handling capabilities can also be used to monitor the process itself by defining metrics and constraints referencing process instance data, e.g. duration of process instance. During the execution the constraints of the processes have to be monitored as long as the according part of the process instance is running. The metrics processing component delivers the input values to the constraint evaluation. Standard engines have to be extended in order to suit those needs, what is discussed in section 5. As realization of this approach we extend the workflow language BPEL with constraint handling capabilities. Therefore as foundation to the further work we provide a formalization parts of the BPEL

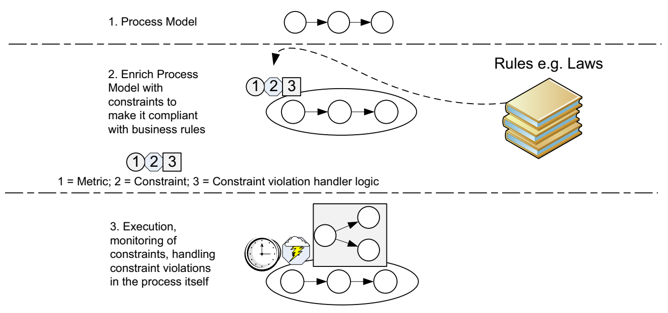
1. Process Model
2. Enrich Process Model with constraints to make it compliant with business rules
Rules e.g. Laws
1 2 3
1 2 3
1 = Metric; 2 = Constraint; 3 = Constraint violation handler logic
3. Execution, monitoring of constraints, handling constraint violations in the process itself

Figure 1. Concept of Constraint Annotations

syntax and semantics. The challenge of this paper lies in extending BPEL with constraint handling capabilities without changing existing semantics in a way that allows to integrate the additional functionalities be implemented in a workflow engine. The paper is organized as follows. To foster the understanding of the paper we provide a short introduction into BPEL (**B**usiness **P**rocess **E**xecution **L**anguage), its formalization and the execution of BPEL processes in section 2. Section 3 introduces the application scenario, which is used to define requirements and to encompass the problem domain. In section 4 we describe the concept provided by this paper followed up with the architecture extending basic workflow engines with enforcement capabilities in section 5. The realization of the scenario with the new concepts is depicted in section 6. To round up this paper we will discuss the related work in section 7. Conclusions are drawn in section 8 including the outlook for further research.

2 BPEL

In this paper we base the realization of the concepts described with BPEL, which can be extended with further functionality. *Business Process Execution Language (BPEL)* is an XML-based workflow language, which is used to describe business processes consisting of several Web services [21]. BPEL's recursive nature allows to provide a business process again as a Web service. Both the process interface and the interfaces of the invoked services are described as WSDL portTypes. These portTypes are referenced via typed connectors called partnerLinks. A BPEL process consists of a set of activities. BPEL defines two types of activities: structured activities and basic activities. The main basic activities allow the process both to invoke Web services (*invoke*) and to be invoked as a Web service (*pick/receive/reply*). All invoke activities refer to a portType which is linked via a partnerLink. Other basic activities are being used for waiting (*wait*), assigning data (*assign*) or doing nothing (*empty*). Structured activities contain other activities and define the business logic between them. The flow activity for instance contains one or more activities, which can be modeled in a graph-oriented way, and therefore be executed in sequence or in parallel. The execution of the enclosed activities can be ordered through the use of control links. Every control link has a Boolean expression called transition condition, which is evaluated at completion on the link's source activity. Every activity being target of links holds a join condition, which is a boolean expression in terms of the incoming link value. Besides flow activities there are other structured activities defined in BPEL like *sequence*, *if-then-else*, *forEach*, *repeatUntil*, and *while*. A *Sequence* executes all activities in the order they are enclosed. The if-then-else activity provides a decision construct. *While*, *repeatUntil*, and *forEach* are used for defining loops. Through the *scope* activity, BPEL provides a facility to encompass a subset of activities and attach certain behaviour by adding fault, compensation, termination or event handlers. A fault handler deals with faults within a scope, whereas a compensation handler is able to reverse successful work done in previous scopes [4]. An event handler processes events on scope or process level and a termination handler provides additional control on terminated scopes.

2.1 A Formalization of BPEL

In this section we provide a formalization of BPEL. We use this formalization later on to define our constraint handling extensions and plug them into BPEL. Therefore we formalize only parts of the abstract BPEL syntax. The complete formalization can be found in [13]. The operational semantics is described in the BPEL 2.0 specification [17]. The general idea of this formalization is to map the XML hierarchy of BPEL into a hierarchy relation HR (e.g. the nesting of activities). All activities contained in a process model are described by the set $\mathcal{A}$. The set $\mathcal{T}_{\mathcal{A}}$ defines the set of all activity types defined in the BPEL specification. Using the function $\mathsf{type}_{\mathcal{A}} : \mathcal{A} \rightarrow \mathcal{T}_{\mathcal{A}}$ each activity is assigned to an activity type. All scope activities of a process model are described using the set $\mathcal{A}_{scopes} \subset \mathcal{A}$. In contrast to the event definition in the BPEL specification, where events are messaging events from the outside, events in this formalization are understood as general events, either internal events created during navigation over the process model or message events sent from the outside. The events are used to trigger the respective handlers. For example, a fault event triggers a fault handler. The hierarchy relation allows handlers to be annotated to activities. The set $\mathcal{E}$ describes all events of a process model. Every event is assigned to an event type. The set $\mathcal{T}_{\mathcal{E}}$ is the set of all event types. The assignment is done using the function $\mathsf{type}_{\mathcal{E}} : \mathcal{E} \rightarrow \mathcal{T}_{\mathcal{E}}$. Events having impact on constraint handling are of the types *event* and *fault*, $\{event, fault\} \subset \mathcal{T}_{\mathcal{E}}$. All events of a process model are contained in the set $\mathcal{E}_{event} \subset \mathcal{E}$. Faults are represented by the set $\mathcal{E}_{fault} \subset \mathcal{E}$.

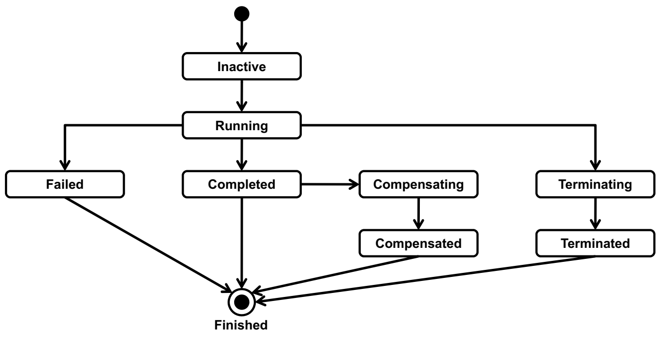

Figure 2. Simplified State Diagram of Scope Activities

2.2 Executing a Scope Activity

In this section we shortly introduce the relevant aspects of the process execution related to our work [14]. Since constraint handling are apply to sets of activities or, in terms of BPEL, *scope* activities, we investigate the execution of a *scope* activity in detail to be able to plug constraint handling capabilities into a scope's execution semantic later on. BPEL processes are executed by navigating through a graph of activities. An activity is started, if all incoming links are evaluated. If the activity is embedded in a structured activity like a *sequence* activity or *if* activity it is started if the parent activity schedules the activity for running and all incoming links are evaluated. The state of an activity changes during its life-cycle. As the *scope* activity is the most relevant activity type for our work, in the following we illustrate its simplified life-cycle (c.f. Fig.2). The BPEL specification does not put any requirements on the implementation and therefore it does not impose any state model for activities. Hence we use a simplified model. The state diagram shown in Figure 2 is based on the models presented in [12], [20], [10] and can be mapped to the most existing BPEL engines (e.g. Apache Ode). Furthermore it abstracts from the death path elimination. Depending on the engine implementation the scope activity might be created either a priori, e.g. on creation of the process instance and set into the state *inactive* or later in the process execution, e.g. when the first incoming link of the scope activity is evaluated and the state set into state *inactive*. The state *inactive* is basically needed to allocate resources necessary for the scopes execution. Once all incoming links are evaluated the activity state is set to *running*, the execution of the activities contained in the scope is started and the event handlers are activated. If all activities and running event handlers finished, the activity state is set to *completed* and the compensation handler is installed. But if a fault happens within the scope while in state *running* the scope gets faulted, which means it changes to state *faulted* and all running activities and scopes encompassed by the faulted scope will terminate. *Terminating* means here to stop all surrounded running activities and scopes and so on. If all activities and scopes are terminated the scope enters the state *terminated* and finishes.

3 Scenario

We will introduce a scenario to illustrate the application of the concept introduced in this paper. The scenario regarded now is located in the area of food safety. Food safety is a very important issue in the production of food and during the delivery of food. There are many laws (e.g. [6]) and quality guidelines (e.g. [3]) which have to be followed and must not be broken for the consumer safety. To ensure the quality of frozen foods the process dealing with the cold chain must be compliant to these rules. A cold chain is a temperature-controlled supply chain, i.e. from a specific point the products must be cooled within a specific temperature range. Cold chains are common in the food and pharmaceutical industries and also some chemical shipments. The basic supply chain of frozen peas is as follows (c.f. figure 3). First, the peas get harvested and as soon as possible transported to a food production fabric. In the fabric the fresh peas must be peeled, washed and blanched. The quick freezing will be done with around -40 $^\circ$C [2]. All these steps have to be done within two days to ensure the quality constraints. If this constraint cannot be hold the peas have to be withdrawn from the whole food production process. After the quick freezing the peas will be packaged and delivered to the customer. A second guideline indicates that after preservation the peas have to be kept at a temperature around -20 to -18 $^\circ$C during all steps of delivery processing till the product is sold to the end consumer. To enforce this guideline a constraint is defined that the surrounding temperature must not exceed -18 $^\circ$C for longer than 2 minutes. If this constraint is violated, an additional quality check is necessary, because the peas are not necessarily spoiled. The check approves either the high quality of the peas, which means that it can be proceeded as usual, or it urges the producer to withdraw the product from the market, due to the high quality of the product could not be kept. The third guideline which must be enforced is the expiry date. If not sold within one year the peas must be withdrawn from the market.

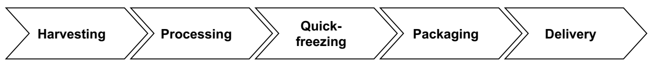

Figure 3. Pea Cold Chain Scenario

4 Integrating Constraint Handling Capabilities into BPEL

In this section we will provide an approach how to extend the existing process metamodel of BPEL with according capabilities to model and monitor constraints and the actions to be executed in case of constraint violation. Therefore new modeling elements are introduced, which are *metric*, *constraint* and *constraint handler*, where constraints define conditions, to be fulfilled. Metrics provide values which serve as input to the parameters used by constraints. Constraint handlers define alternate actions on how to proceed if a constraint violation occurs. To integrate all these modeling elements with their operational semantics into BPEL without changing the BPEL semantics is one of the big challenges of this work. Based on the formalization of BPEL, which we defined in section 2.1 we specify our concepts and how they fit into standard BPEL. But first of all we need to discuss, which way to go to realize constraint handling capabilities in BPEL.

4.1 Discussing realization approaches

In the following we clarify our realization approach. Therefore we discuss the possible ways on how to integrate the new modeling elements in the existing BPEL 2.0 metamodel. Metrics and constraints are affected by process navigation events, e.g. entering a scope event triggers the evaluation and measuring of constraints and metrics. But both metrics and constraints do not affect the process navigation of process model directly. The constraint affects the constraint handler by sending an event message to it, in case of a constraint violation. The task of the constraint handler is to influence the process logic either to terminate the associated scope or to perform additional process logic concurrently. There are basically two possible ways to realize the semantics described above.

- The first way is to define the alternate process logic within the constraint handler.
- The second way is to forward the events received by the constraint handler to already existing modeling elements, which can either be event handlers or fault handlers.

If one inspects the scenario very carefully can find that the actions to be done in reaction to constraint violations are either executing some activities concurrently to the normal execution or terminating the normal execution and performing some compensating activities to limit damage done. This behavior can be realized by using both fault and event handlers. We decide in this paper for the second way using event and fault handlers to realize the desired behavior. Our decision founds on the following points. It's a very comfortable way to realize constraint handling, because the already defined semantics can be reused and must not defined anew. Also this avoids redundancy in semantics and modeling. Constraints handling modeling elements can be annotated to both scopes and activities.

4.2 Defining Constraint Handling Modeling Elements

Constraint handling basically consists of 3 modeling elements. First of all we need to define a metric. Metrics define what shall be measured during the execution of a scope or activity. These metrics can be referenced by every constraint definition which is annotated to an enclosed scope or annotated to the same scope. If a constraint is violated the constraint handler is triggered. Constraint handlers define actions which have to be done on constraint violation. Constraint handlers can be executed either parallel to normal execution or terminating the normal execution.

4.2.1. Metrics. Metrics define values of properties provided by services. Metrics therefore describe exactly the meaning of constraints though they specify how to measure and compute property values respectively. A process model defines a set of metrics $\mathcal{M}$. A metric is annotated on a scope or activity, which forms the metrics annotation relation $\mathcal{MA} \subseteq \mathcal{M} \times \mathcal{A}_{scope}$. Defining a language to express metrics would go beyond the scope of this paper. Hence we refer at this point to [16], where a language to express metrics is defined. The metric evaluation is integrated into the navigation model as follows. After a scope is activated it will be checked whether there are any metrics annotated to it, the annotated metrics get activated, too. Metric processing must be triggered before the first evaluation of the annotated constraints since constraints reference metrics. On process suspension the metric will be deactivated, too.

4.2.2. Constraint. The set of all possible constraints is defined by the Cartesian product $\mathcal{C} \subseteq \mathcal{CC} \times \mathcal{CET}$, where $\mathcal{CC}$ defines a set of constraint conditions and $\mathcal{CET}$ a set of constraint evaluation times with $\mathcal{CET} = \{enter, exit, enter\&exit, continuous\}$. A constraint condition is an expression in first order predicate logic [16], that describes the constraint the scope must meet. It therefore references metrics. The relation is defined through the function $\mathsf{metrics}_{\mathcal{CC}} : \mathcal{CC} \rightarrow 2^{\mathcal{M}}$. Constraint conditions can reference all metrics, which are annotated to a surrounding scope, what means that there can be also metrics referred to which are not annotated to the next surrounding scope but to the parent scope of the surrounding scope, and so on. If the constraint condition evaluates to false the constraint is violated. The constraint evaluation time also called *event quantifiers* specifies, when the predicate must be evaluated relatively to a scopes life cycle. As defined above event quantifiers are *enter*, *exit*, *enter&exit* and *continuous*, which define at what point in time in a scopes life cycle the according constraint condition must be evaluated. E.g. if the evaluation of the constraint is continuous, the first evaluation of the constraint condition in the scope lifetime is done immediately after the scope changed to state *running* (c.f. Figure 2). Before the scope is put into *running* state the constraint remains in state *deactivated*, *unevaluated* (c.f. Figure 4). If the according scope gets into state *running* the constraint state changes to state *activated*, *unevaluated*. Depending on the constraint evaluation time defined the constraint gets evaluated either immediately after *running* constraints with a constraint evaluation time either $\{$*enter*, *enter&exit*, or *continuous*. If the constraint evaluation time of the constraint is set to *exit*, or *enter*, *exit* constraint gets evaluated after all activities including all event handlers an of the scope have completed, which means that scope has entered state *completed*. If a constraint needs to be evaluated more then once in a scope's lif cycle the constraint evaluation time must be set to *continuous*. It might happen that the constraint evaluates differently at those points in time. Hence state transitions between state *valid* and state *violated* state is also a reasonable state transitions (c.f. 4). At any time of the process execution the process might get suspended or resumed. Therefore it must be possible to deactivate or activate the constraints state.

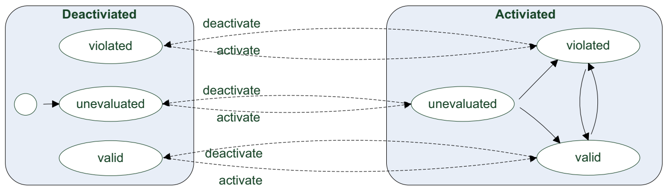

Figure 4. Constraint State Diagram

4.2.3. Handling Constraint Violations. Constraints are annotated at a scope via the constraint handler relation $\mathcal{CH}$ with $\mathcal{CH} \subseteq \mathcal{A}_{scope} \times \mathcal{C} \times \mathcal{E}_{ConstraintAction}$. $\mathcal{E}_{ConstraintAction}$ defines the event message to be generated in case of a constraint violation. This event message can be either of the type simple event message $\mathcal{E}_{event}$ or a fault message, $\mathcal{E}_{fault}$ and $\mathcal{E}_{ConstraintAction} = \mathcal{E}_{event} \cup \mathcal{E}_{fault}$ and $\mathcal{E}_{event}$ and $\mathcal{E}_{fault} \subseteq \mathcal{E}$ with $\mathcal{E}$ are all possible event messages. The type function

$$\mathsf{type}_{\mathcal{E}_{ConstraintAction}}(e) = \begin{cases} message & e \in \mathcal{E}_{event} \\ fault & e \in \mathcal{E}_{fault} \end{cases}$$

enables to find out what's type of the generated event message is. This enables to forward the message to respective handlers, which is either an event handler or a fault handler. The event-types differ in the way how they influence the running process. The constraint violation might violate the rules associated to the scope that seriously, that no successful completion of the scope is possible any more. This is the case where constraint violations are handled as a fault. A fault will be created. This fault message will be handled as any other fault in BPEL processes, terminating all running activities within the scope and performing fault handling as defined in a suitable annotated fault handler. If the constraint violation expresses a less serious breach the process might proceed with its business as usual, but e.g. additional quality checks might be needed as described in chapter 3. This forms the second case the constraint handler will create event messages which can be handled as any other event messages received by the process instance, e.g. with event handlers. Event handlers are executed in a concurrent way to normal business process execution. The constraint handler gets triggered by an event sent by the constraint evaluation component, if a constraint reaches state violated. Therefore a constraint handler might be executed more than once, since less serious constraints might be violated more than once during the execution of the respective scope. Every time the handler might create new event messages. In chapter 3 the constraint, which requires the peas to be cooled at a temperature around -20 to -18 $^\circ$C, might be violated more than once. Therefore every time this happens additional quality checks have to be accomplished, followed by potential withdraw procedures.

5 Runtime Architecture

To enhance a workflow engine with constraint evaluating capabilities it needs to be extended by two further components: a constraint processing component and a metrics managing component [16].

- The metrics managing component provides the values needed by the constraint processing component for the evaluation of the constraints.
- The constraint processing component is responsible to evaluate the constraints at certain points in time.

Both new components need some process navigation events as triggers. For example the metrics managing components needs the process navigation events of a scope with annotated constraint handling capabilities. Especially those events signaling that a scopes enters the *running* or *completed* state, to start or stop the measuring. The process navigation events, which are interesting for constraint processing component, are the same as the ones to trigger the metric measuring component, e.g. enter a scope. If a constraint needs to be evaluated continuously the metrics managing component needs to update corresponding data if the data changes. Hence the metrics managing component signs up for changes for according topic after entering the scope and the first evaluation. Apart from measured data provided by different

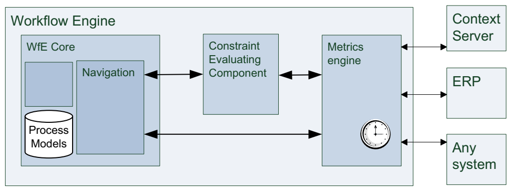

Figure 5. Architecture

systems the metrics managing component has some built-in measures, e.g. time, measured by the core engine or other process metrics. The components identified above can be plugged into an existing workflow engine in two ways, either as internal components to the workflow engine or as external components. We support the idea of integrating the constraint handling components into the workflow engine. Because in our application scenario metrics engine and constraint evaluation component are only used by the workflow engine and need not to be accessible by other outstanding components. Therefore and taking performance aspects into account we recommend an approach with integrated constraint handling capabilities in a workflow engine. The architecture shown in Figure 5 shall only suit our needs regarding the application of metrics and constraint handling in the area of workflows. This architecture can be easily implemented using the pluggable framework described in [11]. The authors in [11] provide a standardized architectural framework for the implementation of extensions to the BPEL language. Implementation of extensions to the BPEL language basically require two interfaces, which enable the extension to react to navigation events and to affect the engines process navigation. Therefore they propose to build a ”generic controller”, which exposes all navigation events to the outside and offers an interface API for incoming events to influence the navigation logic.

6 Example

The implementation of the scenario described in section 3 is shown in figure 6. The scenario is a sequence of coarse grained production steps and is therefore modeled as such. The production steps mapped onto activities are harvesting, followed up the transportation of the harvested goods to the production factory, processing and the freezing of the peas. The frozen peas are packaged, delivered and sold. During execution three constraints have to hold or it must be reacted on their violation. The process has to be finished after a period of time of one year. A constraint and it’s according metric and constraint handler is defined for scope A, which is enclosing all activities. The constraint defining the expiry date of the peas is annotated at this scope and it’s according metric measuring time of each process instance. The activities harvesting up to the freezing of the peas have

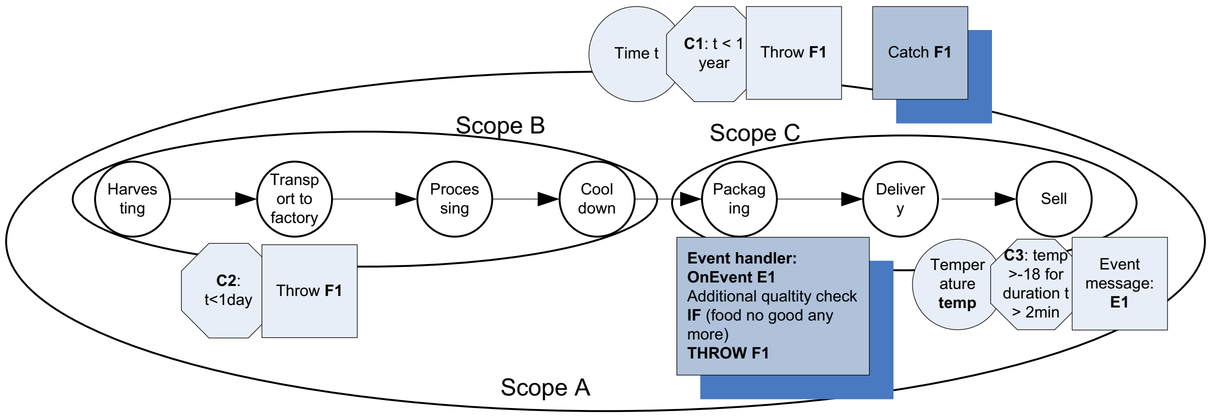

Figure 6. Example - Implementation of Peas Cold Chain Scenario

to be performed during the period of one day. These four activities build scope B together. Because the annotated constraint is also a time constraint and the measurement is not independent from the overall running of the process the metric of scope A can be referenced. If the constraint gets violated a fault F1 is thrown by the constraint handler. The last three activities of the process model are bracketed by the constraint demanding the activities to be performed in following condition. The peas must not have a temperature of -17 and above for the duration of 2 minutes. A metric to measure temperature has to be defined. And again the time metric is referenced. On violation of the constraint handler the very same sends a event message to be received by an event handler. The event handler defines an action to be performed in case of its invocation.

7 Related Work

Todays workflow languages (e.g. BPEL [17]) mostly provide no possibility to explicitly model various constraints and how to handle their violation. Those languages focus on the modeling of the business logic. Just the BPEL language provides appropriate fault and event handling mechanisms [4], to react on events not expected in normal executions. Furthermore some approaches ([18], [5]) break with the fixed modeling of the business logic by modeling the control logic using a set of constraints in contrast to traditional workflow modeling paradigm using e.g. directed graphs or petri nets. [18] uses constraints to describe a partial ordering of activities by defining restrictions on the activity relations. However the aim of the approach presented in this paper is to model and monitor constraints regarding the properties of the environment (e.g. location) and not the properties of activity relations. [19] provides an approach for modeling compliance for business processes. Thereby the authors use rules to check whether a process model is compliant to those rules. Rules are modeled using a Formal Contract Language (FCL), while FCL is based on logic expressions. This approach can be classified as a preventive method to achieve compliance. The approach of our work on the contrary can be classified as reactive method to achieve compliance with certain rules, i.e. we react to constraint violations during runtime. This is necessary e.g. if a workflow has to deal with unforeseen situations running in pervasive environments. Service Level Agreements (SLAs) are used as a contractual basis to define certain properties (e.g. response time) a service (e.g. a business process) has to provide. A SLA also specifies the measures to be taken in case of deviation and failure to meet the asserted service guarantees, e.g. a notification of the service customer. Therefore [16] provides mechanisms to model constraints and metrics. In contrast with our approach violations of constraints are not propagated to the process instance directly. [7] provides an approach for modeling the required message flow of a business process using Linear Temporal Logic (LTL-FO+). However, a violation is not propagated to the process. Traditional WfMS (e.g. [8], [9]) and related standards for human integration [1] allow the definition of escalations for (human) tasks, performed by humans. Escalations allow the specification of constraints. If the constraint is violated a specified action (e.g. a notification) is triggered. In contrast with our approach constraints can only be defined on time and state of a task.

8 Conclusion and Outlook

This paper introduced a new concept on how to explicitly model and enforce constraints on process modeling and process execution respectively. Hence we pointed out a way on how to realize those constraints, including the metrics they refer to, and the actions to be performed in case the constraints get violated in BPEL. We provided an architecture for a workflow engine capable of constraint evaluation and constraint violation handling. Annotating constraints explicitly in business processes enhances flexibility of process models in two ways. We provide a modeling method to integrate business norms into process models. Thereby business logic needs not to be adapted or changed. This simplifies the modeling process, because business norms need not to be translated into business logic, but can stay on its own conceptional level. Additionally business norms might change more often than business logic. Since business norms are attached and not directly integrated in the business logic they can be easily changed. Business processes can be easily reused in different business norm contexts. Furthermore our approach enhances runtime flexibility by means of extending the execution by constraint violation handling actions. Especially in the domain of real world applications this concept provides a useful approach. Context can be considered as constraints that have to hold during a process' execution. The constraint handling framework allows the process to react and adapt the behavior, if context assumptions do not hold during execution of the process, by handling the context change as constraint violation. Less serious constraint violations can be handled through additional tasks, more serious ones through fault handling. Afterwards the processes can proceed with their business as usual. Further research interests in this area are, e.g. to generalize the constraint concept. Constraints might not depend on any metrics. Metrics might be defined as part of the constraint instead. Constraint handling as shown in this paper has basically two options, either to execute additional actions in parallel or to execute alternative actions apart from usual business logic. Other options are possible, e.g. suspend running process and perform additional actions, resume normal process execution.

References

[1] A. Agrawl et al. *Web Services Human Task*. Active Endpoints, Adobe, BEA, IBM, Oracle, SAP AG, June 2007.

[2] H.-D. Belitz, W. Grosch, and P. Schieberle. *Lehrbuch der Lebensmittelchemie*. Springer, Berlin, 2007.

[3] K. Brandriff. *A Focus on Hazard Analysis and Critical Control Points*. United States Department of Agriculture, 2003.

[4] F. Curbera, R. Khalaf, F. Leymann, and S. Weerawarana. Exception Handling in the BPEL4WS Language. In W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, editors, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2003.

[5] H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 25–33, New York, NY, USA, 1998. ACM.

[6] EU. Regulation (ec) no 178/2002 of the european parliament and of the council of 28 january 2002 laying down the general principles and requirements of food law, establishing the european food safety authority and laying down procedures in matters of food safety. *Official Journal of the European Union*, 45:1–24, 2002.

[7] S. Hallé and R. Villemaire. Runtime monitoring of message-based workflows with data. In *EDOC*, pages 63–72. IEEE Computer Society, 2008.

[8] IBM. MQ Series Workflow 3.6.

[9] IBM. WebSphere Process Server 6.1.

[10] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, and F. Leymann. BPEL Event Model. Technical Report Computer Science 2006/10, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, November 2006.

[11] R. Khalaf, D. Karastoyanova, and F. Leymann. Pluggable Framework for Enabling the Execution of Extended BPEL Behavior. In *Proc. of the 3rd ICSOC Int'l Workshop on Engineering Service-Oriented Application: Analysis, Design and Composition (WESOA 2007)*, LNCS. Springer, September 2007.

[12] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. *WS-BPEL Extension for Sub-processes – BPEL-SPE*. IBM, SAP, 2005.

[13] O. Kopp, R. Mietzner, and F. Leymann. Abstract Syntax of WS-BPEL 2.0. Technical Report Computer Science 2008/06, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, September 2008.

[14] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall, Upper Saddle River, New Jersey, 2000.

[15] D. Lucke, C. Constantinescu, and E. Westkämper. Smart factory - a step towards the next generation of manufacturing. In *Manufacturing Systems and Technologies for the New Frontier*, 2008.

[16] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. *Web Service Level Agreement (WSLA) Language Specification*. IBM, 2003.

[17] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language Version 2.0*, Mar 2007.

[18] M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst. Constraint-based workflow models: Change made easy. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer, 2007.

[19] S. W. Sadiq, G. Governatori, and K. Namiri. Modeling Control Objectives for Business Process Compliance. In G. Alonso, P. Dadam, and M. Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 149–164. Springer, 2007.

[20] T. Steinmetz. Ein Event-Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE. Diploma thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, August 2008.

[21] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture*. Prentice Hall, April 2005.

[22] K. Windt and M. Hülsmann. *Understanding Autonomous Cooperation & Control in Logistics: The Impact on Management, Information, Communication and Material Flow*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.