Real-Time Risk Monitoring in Business Processes: A Sensor-based Approach

Raffaele Conforti[a], Marcello La Rosa[a,b], Giancarlo Fortino[c], Arthur H. M. ter Hofstede[a,b,d], Jan Recker[a]

[a] *Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia*
E-mail: {raffaele.conforti,m.larosa,a.terhofstede,j.recker}@qut.edu.au
[b] *NICTA Queensland Lab, PO Box 6020, St Lucia QLD 4067, Australia*
[c] *Universita' della Calabria, Via P. Bucci, cubo 41C, 87036 Rende (CS), ITALY*
E-mail: g.fortino@unical.it
[d] *Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

Abstract

This article proposes an approach for real-time monitoring of risks in executable business process models. The approach considers risks in all phases of the business process management lifecycle, from process design, where risks are defined on top of process models, through to process diagnosis, where risks are detected during process execution. The approach has been realized via a distributed, sensor-based architecture. At design-time, sensors are defined to specify risk conditions which when fulfilled, are a likely indicator of negative process states (*faults*) to eventuate. Both historical and current process execution data can be used to compose such conditions. At run-time, each sensor independently notifies a sensor manager when a risk is detected. In turn, the sensor manager interacts with the monitoring component of a business process management system to prompt the results to process administrators who may take remedial actions. The proposed architecture has been implemented on top of the YAWL system, and evaluated through performance measurements and usability tests with end users. The experiments show that risk conditions can be computed efficiently and that the approach is perceived as useful by the users.

1. Introduction

Business processes are constantly exposed to a wide range of risks. As demonstrated by the recent incidents in the finance sector (the 4.9B Euros fraud at Société Générale), in the health sector (Patel Inquiry) and in the aviation industry (terrorist attacks), failures of process-driven risk management can result in substantial financial and reputational consequences, potentially threatening an organization's existence.

According to the AS/NZS ISO 31000 standard, a *business process risk* is the chance of something happening that will have an impact on the process objectives, and is measured in terms of likelihood and consequence [69]. Legislative initiatives such as the Sarbanes-Oxley Act[1] and Basel II [6] in the finance sector have highlighted

[1] `www.gpo.gov/fdsys/pkg/PLAW-107publ204`

the pressing need to better manage business process risks. As a consequence of these mandates, organizations are now seeking new ways to *control* process-related risk and are attempting to incorporate it as a distinct view in their operational management. However, whilst conceptually appealing, to date there is little guidance as to how this can be done concretely. Currently, the disciplines of process management and risk management are largely disjoint and operate independently of one another. In industry they are usually handled by different organizational units. Within academia, recent research has centered on the characterization of process-related risks. However the incidents described above show that a focus on risk analysis alone is no longer adequate, and an active, real-time risk detection is required.

We propose a concrete approach for real-time monitoring of risks in executable business process models. This is achieved by embedding risks into all phases of the BPM lifecycle: from process design, where high-level risks defined via a risk analysis method are mapped down to specific process model elements such as activities, resources and data, through to process diagnosis, where risks are detected during process execution. By automating risk detection, the interested users (e.g. a process administrator) can be notified as early as a risk is detected (i.e. in real-time), such that remedial actions can be taken to rectify the current process instance, and prevent an undesired state of the process (*fault* for short), from occurring. Based on historical data, we can also compute the probability of a risk at run-time, and compare it to a threshold, so as to notify the user only when the risk is no longer tolerable.

The proposed approach is operationalized via a distributed, sensor-based architecture on top of Business Process Management Systems (BPMSs). Each sensor is coupled with a risk condition capturing the situation upon which the risk of a given fault may occur. Sensors are defined at design-time on the executable process model. Conditions can be determined via a query language that can fetch both historical and current execution data from the logs of the BPMS. At run-time sensors are registered with a central sensor manager. At a given sampling rate, or based on the occurrence of a specific event, the sensor manager retrieves and filters all data relevant for the various sensors (as it is logged by the BPMS engine), and distributes it to the relevant sensors. If a sensor condition holds, i.e. if the probability of the associated risk is above a given threshold, the sensor alerts the sensor manager which in turn notifies the monitoring component of the BPMS. The distributed nature of the architecture guarantees that there is no performance overhead on the BPMS engine, and thus on the execution of the various process instances. We implemented this architecture on top of the YAWL system. We extended the YAWL Editor to cater for the design of risk sensors, and equipped the run-time environment with a sensor manager service that interacts with YAWL's monitoring service and execution engine. Finally, to facilitate the definition of process risks, we implemented a set of risk templates for various categories, such as organizational, data-related and technology-related. Such templates are abstract risk definitions which users need to bind to concrete process elements.

To prove the feasibility of the proposed approach, we used fault tree analysis [12] (a well-established risk analysis method) to

identify risk conditions in a reference process model for logistics, in collaboration with an Australian risk consultant. These risks embrace different process aspects such as tasks' order dependencies, involved resources and business data, and relate to historical data where needed, to compute risk probabilities. We expressed these conditions via sensors in the YAWL system, and measured the time needed to compute these conditions at run-time. The tests showed that the sensor conditions can be computed in a matter of milliseconds without impacting on the performance of the running process instances. Finally, we conducted a usability analysis of the approach and its implementation, by administering an online questionnaire to users of the YAWL system. The results of this analysis showed that the approach is perceived as being interesting and useful, confirming an intention of the participants to use it.

This paper is organized as follows. Section 2 illustrates the running example in the logistics domain. Section 3 describes our risk-aware BPM approach while Section 4 presents the sensor-based architecture to implement this approach. Section 5 formalizes the abstract syntax of the language proposed for risk detection while Section 6 shows how the risks defined on the running example can be modeled via this language. Next, Section 7 proposes a set of risk templates to facilitate the definition of process risks. Section 8 illustrates the implementation of the sensor-based architecture, while Section 9 reports on the results of the performance and usability evaluations. Section 10 covers related work while Section 11 concludes the paper. Appendix A provides a short description of the actions defined in the language. Appendix B describes the nested loops that can be executed during the verification of a sensor condition while Appendix C describes the functions that can be invoked on variables (that are resources) during the verification of a sensor condition. Finally, Appendix D provides the questionnaire used in the usability experiment.

2. Running Example

In this section we use an example to illustrate how the risk of possible faults to occur during a business process execution can be identified as early as possible. In particular, we show how risks can be expressed in terms of process-specific aspects such as tasks occurrence, data or available resources. Figure 1 describes the payment subprocess of an order fulfillment business process which is inspired by the VICS industry standard for logistics [77]. The notation used to represent this example is that of YAWL [74], although a deep knowledge of this language is not required.

This process starts after the freight has been picked up by a carrier and deals with the shipment payment. The first task is the production of a Shipment Invoice containing the shipment costs related to a specific order for a specific customer. If shipments have been paid in advance, all that is required is for a Finance Officer to issue a Shipment Remittance Advice specifying the amount being debited to the customer. Otherwise, the Finance Officer issues a Shipment Payment Order that needs to be approved by a Senior Finance Officer (who is the superior of this Finance Officer). At this point, a number of updates may be made to the Shipment Payment Order by the Finance Officer that issued it, but each of these needs to be approved by the Senior Finance Officer. After the docu-

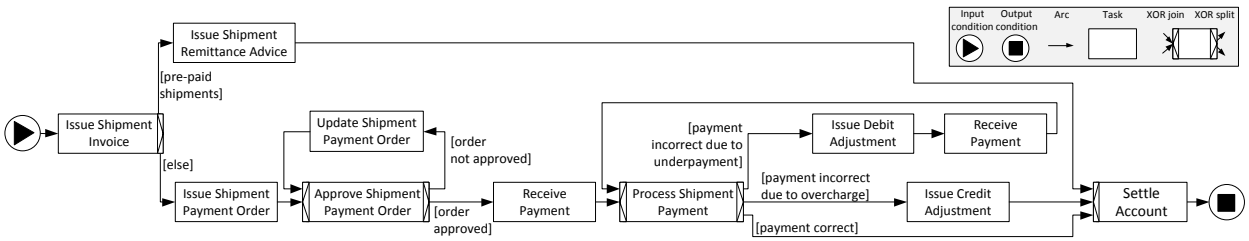


Figure 1: Order-Fulfillment: Payment subprocess.

ment is finalized and the customer has paid, an Account Manager can process the shipment payment by specifying the balance. If the customer underpaid, the Account Manager needs to issue a Debit Adjustment, the customer needs to pay the balance and the payment needs to be reprocessed. A customer may also overpay. In this case the Account Manager needs to issue a Credit Adjustment. In the latter case and in case of a correct payment, the shipment payment process is completed.

In collaboration with a risk analyst of an Australian consulting company, we identified four faults that can occur during the execution of this payment subprocess. In order to prevent the occurrence of these faults, for each of them we also defined an associated risk condition by using fault tree analysis [12]. Accordingly, each risk condition is expressed as a set of lower-level boolean events which are organized in a tree via logical connectives such as ORs, ANDs and XORs.

The first fault is an *overtime process* fault. A Service Level Agreement (SLA) for a process or for a given task within a process, may establish that the process (or task) may not last longer than a Maximum Cycle Time MCT, otherwise the organization running the process may incur a pecuniary penalty. In our case, an overtime fault occurs if an instance of the payment subprocess is not completed within an MCT of five days.

To detect the risk of overtime fault at run-time, we should check the likelihood that the running instance does not exceed the MCT based on the amount of time T_c expired at the current stage of the execution. Let us consider T_e as the remaining cycle time, i.e. the amount of time estimated to complete the current instance given T_c. Then the probability of exceeding MCT can be computed as $1 - MCT/(T_e + T_c)$ if $T_e + T_c > MCT$ and is equal to 0 if $T_e + T_c \leq MCT$. If this probability is greater than a tolerance value (e.g. 60%), we notify the risk to the user. The estimation of the remaining cycle time is based on past executions of the same process and can be computed using the approach in [76] (see Section 9 for more details).

The second fault is related to the resources participating in the process. The Senior Finance Officer who has approved a Shipment Payment Order for a given customer, must have not approved another order by the same customer in the last d days, otherwise there is an *approval fraud*. This fault is thus generated by the violation of a four-eye principle across different instances of the Payment subprocess.

To detect the risk of this fault we first have to check that there is an order, say order o of customer c, to be approved. This

means checking that an instance of task Approve Shipment Payment Order is being executed. Moreover, we need to check that either of the following conditions holds: i) o has been allocated to a Senior Finance Officer who has already approved another order for the same customer in the last d days; or ii) at least one Senior Finance Officer is available who approved an order for customer c in the last d days and all other Senior Finance Officers who never approved an order for c during the last d days are available. The corresponding fault tree is shown in Figure 2.

The third fault relates to a situation where a process instance executes a given task too many times. This situation typically occurs in the context of loops. Not only could this lead to a process slowdown but also to a "livelock" if the task is in a loop whose exit condition is purposefully never met. In general, given a task t a maximum number of allowable executions of t per process instance $MAE^i(t)$ can be fixed as part of the SLA for t. With reference to the Payment subprocess, this can occur for example if task Update Shipment Payment Order is re-executed five times within the same process instance. We call this an *order unfulfillment* fault.

To detect the risk of this fault at runtime, we need to check if: i) an order o is been updated (i.e. task Update Shipment Payment Order is currently being performed for order o); and ii) it is likely that this order will be updated again (i.e. task Update Shipment Payment Order will be repeated within the same process instance). The probability that the number of times a task will be repeated within the same instance of the Payment subprocess is computed by dividing the number of instances where the MAE^i for task Update Shipment Payment Order has been reached, over the number of instances that have executed this task at least as many times as it has been executed by the current instance, and have completed. The tolerance value indicates a threshold above which the risk should be notified to the user. For example, if this threshold is 60% for task t, a risk should be raised if the probability of $MAE^i(t)$ is greater than 0.6.

The fourth fault is an *underpayment fraud*. It relates to a situation in which a given task is executed too many times across multiple process instances. Similar to the previous fault, given a task t we can define a maximum number of allowable executions of t per process $MAE^p(t)$ as part of the SLA for p. In our example, this type of fault occurs when a customer underpays more than three times within the last five days.

To detect the risk of underpayment fraud, we need to check if: i) a debit adjustment is currently being issued to a customer c (i.e. task Issue Debit Adjustment is currently being performed for customer c); and ii) it is likely that the maximum number of debit adjustments will be issued to the same customer in a d-day time frame. The probability that MAE^p is reached for task Issue Debit Adjustment of customer c in d days is computed by dividing the number of customers for which the MAE^p for task Issue Debit Adjustment has been reached within d days, over the number of customers for which this task has been executed at least as many times as it has been executed for c within d days. If this probability is above a tolerance value, the risk should be raised and the user notified. Similar to the previous risk, the tolerance value indicates a threshold above which this risk should be notified to the user. The corresponding fault-tree is shown in Figure 2.

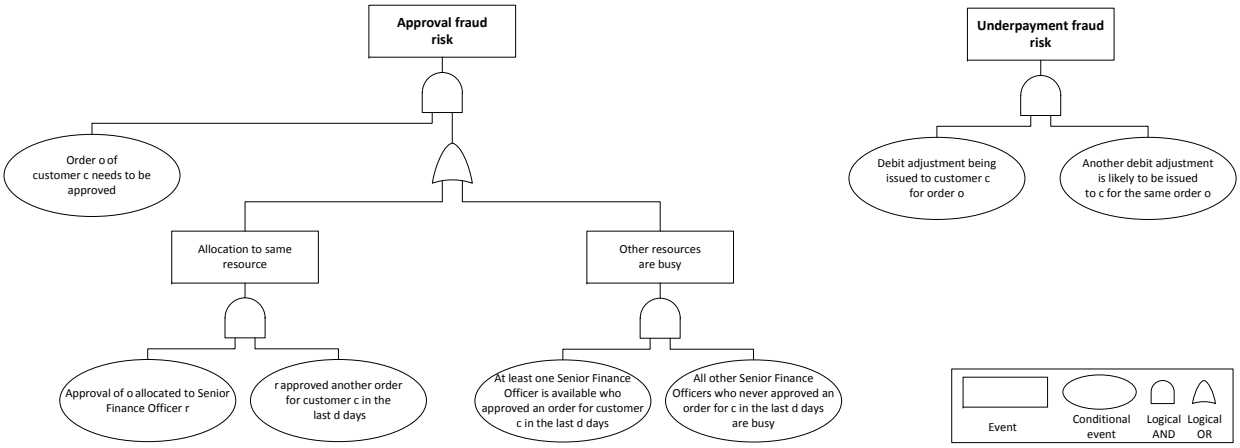


Figure 2: The fault trees for Approval Fraud and Underpayment Fraud.

3. Risk-aware Business Process Management

As we have seen in the context of the payment example, a fault in a business process is an undesired state of a process instance which may lead to a process failure (e.g. the violation of a policy may lead to a process instance being interrupted). Identifying a fault in a process requires determining the condition upon which the fault occurs. For example, in the payment subprocess, we have an underpayment fraud if a customer underpays more than three times within a five-day time frame.

However, a *fault condition* holds only when the associated fault has occurred, which is typically too late to avoid a process failure. Indeed, we need to be able to estimate the risk of a process fault, i.e. if, and possibly with what likelihood, the fault will occur in the future. Early risk detection allows process users to promptly react with countermeasures, if any, to prevent the related fault from occurring at all.

We use the notion of *risk condition*, as opposed to fault condition, to describe the set of events that lead to the possibility of a fault to occur in the future. In order to evaluate risk conditions "on-line", i.e. while a process instance is being executed, we need to consider the current state of the BPMS. This means knowing the state of all running instances of any process (and not only the state of the instance for which we are computing the risk condition), the resources that are busy and those that are available, and the values of the data variables being created and consumed. Moreover, we need to know the historical data, i.e. the execution data of all instances that have been completed. In particular, we can use historical data to estimate the probability of a given fault to occur, i.e. the *risk probability*. For example, for the underpayment fraud, we can estimate the likelihood that another debit adjustment is being issued for a given combination of customer/order (historical data), given that one such debit adjustment has just been issued (current data). To obtain a boolean risk condition, we compare the risk probability that we obtain with a tolerance value, such that the condition holds if the risk probability exceeds the given threshold. For example, we raise the risk of underpayment fraud if the

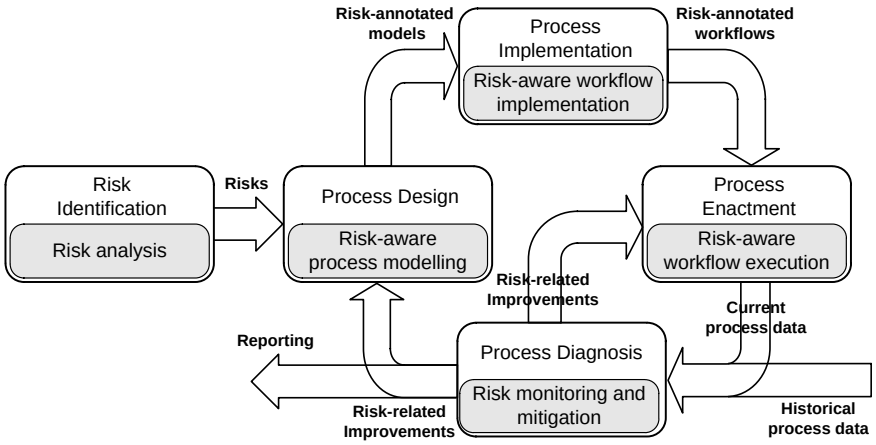


Figure 3: Risk-aware Business Process Management lifecycle.

risk probability is greater then 60%.

In other cases, we may avoid to embed a risk probability in the risk condition, if we are able to determine the occurrence of a set of events which directly leads to a high risk. This is the case of the approval fraud, where both the events "Allocation to same resource" and "Other resources are busy" already signal a high risk of approval fraud.

Based on these considerations, we present a novel approach for on-line risk detection in business processes. The focal idea of this approach, shown in Figure 3, is to embed elements of risk into all four phases of the traditional BPM lifecycle [21].

Input to this "risk-aware" BPM lifecycle is a *Risk Identification* phase, where risk analysis is carried out to identify risks in the process model to be designed. Traditional risk analysis methods such as FTA (as seen in the previous section), Root Cause Analysis [35] or CORAS [43], can be employed in this phase. The output of this phase is a set of risks, each expressed as a risk condition.

Next, in the *Process Design* phase, these high-level risk conditions are mapped down to process model-specific aspects. For example, the condition "debit adjustment being issued to customer c for order o" is mapped to the occurrence of a specific task, namely "Issue Debit Adjustment" in the Payment process model. The result of this second phase is a risk-annotated process model. In the next phase, *Process Implementation*, these conditions are linked to workflow-specific aspects, such as content of variables, and resource allocation states. For example, "customer c" is linked to the `Customer` element of the XML representation of the `Debit Adjustment` document. Process Implementation may be integrated with Process Design if the language used at design-time is executable (e.g. BPMN 2.0 or YAWL).

The risk-annotated workflow model resulting from Process Implementation is then executed by a risk-aware process engine during *Process Enactment*. Historical data stored in process logs, and current execution data coming from process enactment, are filtered, aggregated and analyzed in the *Process diagnosis* phase, in order to evaluate the various risk conditions. When a risk condition evaluates to true, the interested users (e.g. a process administrator) are notified and reports can also be produced during

this phase for auditing purposes. Finally, this phase can trigger changes in the current process instance, to mitigate the likelihood of a fault to occur, or in the underlying process model, to prevent a given risk from occurring ever again.

In the next section we describe a sensor-based architecture to operationalize this enhanced BPM lifecycle.

4. Sensor-based Realization

In order to realize our risk-aware BPM lifecycle, we devised an approach based on sensors. In a nutshell, the idea is to capture risk and fault conditions via sensors, and then monitor these sensors during process execution. An overview of this approach is shown in Figure 4 using the BPMN 2.0 notation [50].

Sensors are defined during the *Process Design* and *Process Implementation* phases of our risk-aware BPM lifecycle (see Figure 3), for each process model for which the presence of risks and/or faults need to be monitored. If the process model is specified via an executable language, then these two phases coincide.

A sensor is defined through a boolean *sensor condition*, constructed on a set of process variables, and a *sensor activation trigger*. Process variables are used to retrieve information from the specific instance in which the sensor condition will be evaluated as well as from other instances, either completed or still running. For example, we can use variables to retrieve the resource allocated to a given task, the value of a task variable, or the status of a task. Process instances can either be identified based on the current instance (e.g. the last five instances that have been completed before the current one), or based on the fulfillment of a *case condition* (e.g. "all instances where a given resource has executed a given task"). The sensor condition can represent either a *risk condition* associated with a fault, or a *fault condition*, or both. If both conditions are specified, the fault condition is evaluated only if the risk condition evaluates to true. For example, the sensor will check if an overtime process fault has occurred in a process instance only if first the risk of such fault has first been detected, based on the estimation of the remaining cycle time for this instance. Finally, the sensor activation trigger can be either a timer periodically fired according to a sampling rate (e.g. every 5 minutes), or an event emitted by the process engine (e.g. the completion of a task).

During *Process Enactment*, the defined sensors are registered with a *sensor manager*, which activates them. In the *Process Diagnosis* phase, which starts as soon as the process is enacted, the activated sensors receive updates on the variables of their sensor conditions according to their trigger (timer or event). When a sensor receives an update, it checks its sensor condition. If the condition holds, a notification is sent from the sensor to the monitor service of the BPMS.

The sensor manager relies on three interfaces to interact with the BPMS (see Figure 5(a)):

- *Engine interface*, used to register a sensor with a particular event raised by the BPMS engine. When the event occurs the sensor is notified by the sensor manager.

- *Database interface*, used to query the BPMS database in order to collect current and historical information.

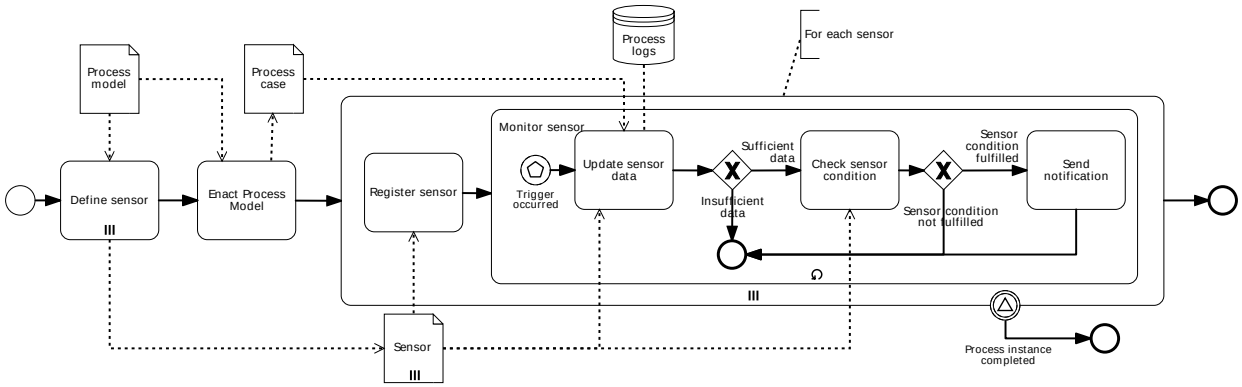


Figure 4: Realization of risk-aware BPM lifecycle via sensors.

- *Monitor interface*, used to notify the detection of risks and faults to the monitor service of the BPMS.

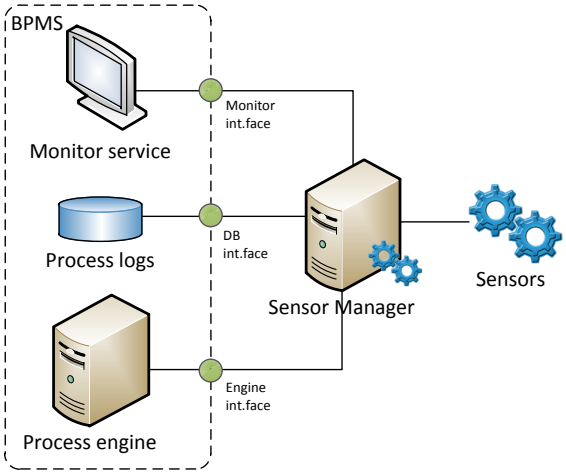
These interfaces can be implemented by the vendor or user of the BPMS where the sensor manager needs to be installed. In this way, our sensor manager can virtually be interfaced with any BPMS. As an example, the conceptual model of the database interface is showed in Figure 5(b), where methods have been omitted for space reasons. This conceptual model is inspired by the reference process meta-model of the WfMC [31], in order to cover as many aspects as possible of a workflow model, and meantime, to remain as generic as possible. For example, class *WorkFlowDefinition* allows one to retrieve information about the process model where the sensor is defined, such as process identifier and name, while class *SubProcess* allows one to retrieve information about a specific subprocess, and so on. This interface should be implemented according to the characteristics of the specific database used in the BPMS at hand. For an efficient use of the interface, one should also define indexes on the attributes of the BPMS database that map the underlined attributes in Figure 5(b). These indexes have been determined based on the types of queries that can be defined in our sensor definition language.

An alternative approach to achieve the portability of the sensor manager, would be to read the BPMS logs from a standard serialization format such as OpenXES [26]. However, as we will show in Section 9, this solution is rather inefficient.

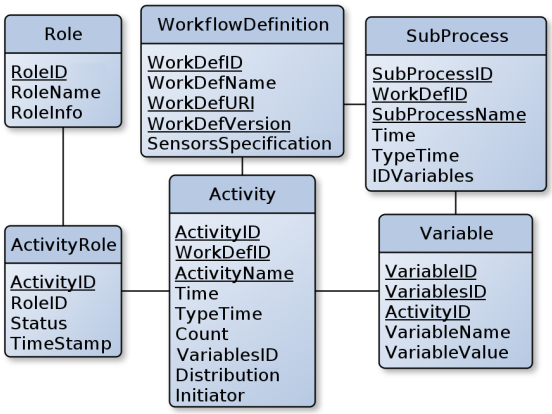
The advantages of using sensors are twofold. First, their conditions can be monitored while the process model is being executed, i.e. in real-time. Second, according to a distributed architecture, each sensor takes care of checking its own condition after being activated by the sensor manager. In this way, potential execution slowdowns are avoided (e.g., the process engine and the sensor manager could be deployed onto two different machines).

5. Sensor Definition Language: Abstract Syntax

In the following, the sensor definition language is explained in detail via an abstract syntax [47]. The definition of a sensor requires a *risk condition*, a boolean *fault condition*, a *sensor activation trigger*, and a *consequence* which represents the gravity of



(a)



(b)

Figure 5: Sensor-based architecture (a); Database Interface schema model (b).

the impact the fault will have on the company in case it occurs. The trigger can be a timer based on a sampling rate, or an event produced by the engine interface. Risk condition and fault condition are constructed on a set of process variables.

$$
\begin{aligned}
\mathit{Sensor} &\triangleq v : \mathit{Variables}; t : \mathit{Trigger}; \\
&\quad \mathit{riskCond} : \mathit{RiskCond}; \\
&\quad \mathit{faultCond} : \mathit{BoolCond}; \\
&\quad \mathit{consequence} : \mathit{MaCond}; \\
\mathit{Trigger} &\triangleq \mathit{timer} \mid \mathit{event}
\end{aligned}
$$

A variable is defined with a mapping among a *varName* and an *Information*. This *Information* can be a constant (using *Definition*), or the result of a function executed on a variable (using *VarFunc*), or a piece of information collected from the process instance (using *CaseExp* or *CaseEleExp*). We use a *CaseExp* if the information is related to the process instance itself, while a *CaseEleExp* if the information is related to an element of a process instance (that must be specified using *TaskOrNet* that identifies a task by taskLabel or a net by netName).

$$
\begin{aligned}
\mathit{Variables} &\triangleq \mathit{Assignment}^{+} \\
\mathit{Assignment} &\triangleq \mathit{result} : \mathit{varName}; \\
&\quad i : \mathit{Information} \\
\mathit{Information} &\triangleq \mathit{Definition} \mid \mathit{VarFunc} \mid \\
&\quad \mathit{CaseExp} \mid \mathit{CaseEleExp} \\
\mathit{VarFunc} &\triangleq \mathit{ResCond} \mid \mathit{ResSimFunc} \mid \\
&\quad \mathit{ResComFunc} \\
\mathit{Definition} &\triangleq c : \mathit{constant} \\
\mathit{CaseEleExp} &\triangleq \mathit{ce} : \mathit{CaseExp}; \\
&\quad \mathit{ton} : \mathit{TaskOrNet} \\
\mathit{TaskOrNet} &\triangleq \mathit{taskLabel} \mid \mathit{netName}
\end{aligned}
$$

When we use a *CaseExp* we must specify the instances of interest (using *CaseIDStat*), and the action that identifies the piece of information (using *Action*). Such *Action* can either be an information related to a predicate function, a predicate function with input, task or net variable, or a task or net subvariable. An instance can be identified in various ways, by its position among all the instances of the same process model (using *absExp*), by its position respect the current instance (using *relExp*), or by the fulfillment of some conditions (us-

ing *CaseCondSet*).

$$
\begin{array}{rcl}
\mathit{CaseExp} & \triangleq & \mathit{cis} : \mathit{CaseIDStat}; a : \mathit{Action} \\
\mathit{CaseIDStat} & \triangleq & \mathit{absExp} \mid \mathit{relExp} \mid \\
 & & \mathit{CaseCondSet} \\
\mathit{Action} & \triangleq & \mathit{predFunc} \mid \mathit{taskOrNetVar} \mid \\
 & & \mathit{SubVarExp} \mid \mathit{inputPredFunc} \\
\mathit{SubVarExp} & \triangleq & \mathit{var}^{+}
\end{array}
$$

These conditions can be on the identifier of the process instance (using *CaseParam*), or on a piece of information related to a task or a net (using *CaseCond*). It is also possible to specify multiple conditions that are obtained by the conjunction of several *CaseParam* and *CaseCond* elements (using *CaseCondExp*). Whether using a *CaseParam* or a *CaseCond* it will be compared with a *RHandExp*. A *RHandExp* can be a *constant*, a *function*, a *varName*, or an expression containing those elements.

$$
\begin{array}{rcl}
\mathit{CaseCondSet} & \triangleq & \mathit{CaseCond} \mid \mathit{CaseParam} \mid \\
 & & \mathit{CaseCondExp} \\
\mathit{CaseCondExp} & \triangleq & \mathit{ccs}_1, \mathit{ccs}_2 : \mathit{CaseCondSet}; \\
 & & \mathit{bo} : \mathit{BoolOp} \\
\mathit{CaseCond} & \triangleq & \mathit{ton} : \mathit{TaskOrNet}; \\
 & & a : \mathit{Action}; \mathit{co} : \mathit{CompOp}; \\
 & & \mathit{rhe} : \mathit{RHandExp} \\
\mathit{CaseParam} & \triangleq & i : \mathit{idFunc}; \mathit{co} : \mathit{CompOp}; \\
 & & \mathit{rhe} : \mathit{RHandExp} \\
\mathit{CompOp} & \triangleq & \mathit{le} \mid \mathit{leq} \mid \mathit{ge} \mid \mathit{geq} \mid \mathit{eq} \mid \\
 & & \mathit{contains} \mid \mathit{isContained} \\
\mathit{RHandExp} & \triangleq & \mathit{constant} \mid \mathit{function} \mid \\
 & & \mathit{varName} \mid \mathit{RHandExpSet} \\
\mathit{RHandExpSet} & \triangleq & \mathit{rhe}_1, \mathit{rhe}_2 : \mathit{RHandExp}; \\
 & & o : \mathit{Operator}
\end{array}
$$

Once all the variables have been specified the risk condition can be defined. A *RiskCond* is composed of two *MaCond* elements, one is the risk likelihood and the other is the risk threshold. A risk condition evaluates true if the likelihood exceeds the threshold. A *MaCond* is a arithmetical expression that can use constants, variables, results of functions invoked on variables (using *ResSimFunc*, and *ResComFunc*), and the results obtained by the execution of loops (using *MaFor*, and *FixMaFor*). A *MaCond* may be in a conditional form, represented as *MaITE*, or be a normal expression represented as *MaExp*, despite it is always possible to have conditional elements inside a normal expression. A conditional expression *MaITE* is composed of a *BoolCond* representing the *if* and two *MaCond*s representing respectively the *then* and *else*.

$$
\begin{array}{rcl}
\mathit{RiskCond} & \triangleq & \mathit{riskL}, \mathit{riskT} : \mathit{MaCond} \\
\mathit{MaCond} & \triangleq & \mathit{MaITE} \mid \mathit{MaExp} \mid \\
 & & \mathit{MaFor} \mid \mathit{FixMaFor} \mid \\
 & & \mathit{ResSimFunc} \mid \mathit{constant} \mid \\
 & & \mathit{ResComFunc} \mid \mathit{varName} \\
\mathit{MaITE} & \triangleq & \mathit{if} : \mathit{BoolCond}; \\
 & & \mathit{then}, \mathit{else} : \mathit{MaCond} \\
\mathit{MaExp} & \triangleq & \mathit{MaUnExp} \mid \mathit{MaBinExp} \\
\mathit{MaUnExp} & \triangleq & s : \mathit{sub}; \mathit{me} : \mathit{MaCond} \\
\mathit{MaBinExp} & \triangleq & \mathit{me}_1, \mathit{me}_2 : \mathit{MaCond}; \\
 & & \mathit{mo} : \mathit{MaOp} \\
\mathit{MaOp} & \triangleq & \mathit{add} \mid \mathit{sub} \mid \mathit{mul} \mid \mathit{div} \mid \mathit{exp} \mid \\
 & & \mathit{mod}
\end{array}
$$

As said before a condition expression is defined using a *BoolCond*. A *BoolCond* is a boolean expression that can use constants, variables, results of functions invoked on variables (using *ResCond*), and the results obtained by the execution of loops (using *BoolFor*, and *FixBoolFor*). A *BoolCond* may be in a conditional form, represented as *BoolITE*, or be a normal expression represented as *BoolExp*, despite it is always possible to have conditional elements inside a normal expression. A conditional expression *BoolITE* is composed of three *BoolCond*s representing the *if*, the *then*, and the *else*. From the definition of the *BoolExp* element on, the syntax describes a boolean expression. In fact a *BoolExp* element can be solved via the analysis of an unary expression *BoolUnExp* or a binary expression

BoolBinExp, that can contain the result of a comparison *Comp*, and so on.

$$
\begin{array}{rcl}
\mathit{BoolCond} & \triangleq & \mathit{BoolITE} \mid \mathit{BoolExp} \mid \\
 & & \mathit{BoolFor} \mid \mathit{FixBoolFor} \mid \\
 & & \mathit{Comp} \mid \mathit{ResCond} \mid \\
 & & \mathit{varName} \mid \mathit{constant} \\
\mathit{BoolITE} & \triangleq & \mathit{if}, \mathit{then}, \mathit{else} : \mathit{BoolExp} \\
\mathit{BoolExp} & \triangleq & \mathit{BoolUnExp} \mid \mathit{BoolBinExp} \\
\mathit{BoolUnExp} & \triangleq & n : \mathit{neg}; e : \mathit{BoolCond} \\
\mathit{BoolBinExp} & \triangleq & e_1, e_2 : \mathit{BoolCond}; \\
 & & \mathit{bo} : \mathit{BoolOp} \\
\mathit{BoolOp} & \triangleq & \mathit{and} \mid \mathit{or} \\
\mathit{Comp} & \triangleq & \mathit{ce}_1, \mathit{ce}_2 : \mathit{CompElem}; \\
 & & \mathit{co} : \mathit{CompOp} \\
\mathit{CompElem} & \triangleq & \mathit{MaCond} \mid \mathit{ResListFunc} \mid \\
 & & \mathit{varName} \mid \mathit{constant} \\
\mathit{CompOp} & \triangleq & \mathit{lt} \mid \mathit{lteq} \mid \mathit{eq} \mid \mathit{gteq} \mid \mathit{gt} \mid \\
 & & \mathit{noteq}
\end{array}
$$

The functions that can be invoked on a variable are identified by the elements: *ResCond*, *ResListFunc*, *ResSimFunc*, and *ResComFunc*. This elements can be used only in specific points of the syntax since the result returned can be a boolean, or a list, or a number.

$$
\begin{array}{rcl}
\mathit{ResCond} & \triangleq & \mathit{res} : \mathit{varName}; a : \mathit{Activity} \\
\mathit{ResListFunc} & \triangleq & \mathit{result} : \mathit{varName}; \\
 & & \mathit{plrf} : \mathit{predListResFunc} \\
\mathit{ResSimFunc} & \triangleq & \mathit{resource} : \mathit{varName}; \\
 & & \mathit{psrf} : \mathit{predSimpleResFunc} \\
\mathit{ResComFunc} & \triangleq & \mathit{res}_1, \mathit{res}_2 : \mathit{varName}; \\
 & & \mathit{pcrf} : \mathit{predComplResFunc} \\
\mathit{Activity} & \triangleq & \mathit{resource} : \mathit{varName}; \\
 & & \mathit{lrf} : \mathit{ResListFunc}
\end{array}
$$

The constructs *MaFor* and *BoolFor* are used to execute nested loops. The definition of one of these elements requires to specify the type of loop (*TypeBF* for *BoolFor* or *TypeMF* for *MaFor*), the list of variables that will be used to create the nested loops (one loops for each variable), and the expression that must be executed. The *TypeBF* and *TypeMF* describe how the results of each execution will be joined, it is possible to choose among four types of loops: and (i.e. logic AND), or (i.e. logic OR), add (i.e. addition), and mul (i.e. multiplication).

$$
\begin{array}{rcl}
\mathit{MaFor} & \triangleq & t : \mathit{TypeMF}; \mathit{lr} : \mathit{ListResult}; \\
 & & \mathit{fme} : \mathit{ForMaCond} \\
\mathit{FixMaFor} & \triangleq & \mathit{fixEle} : \mathit{varName}; \\
 & & \mathit{mf} : \mathit{MaFor} \\
\mathit{TypeMF} & \triangleq & \mathit{add} \mid \mathit{mul} \\
\mathit{BoolFor} & \triangleq & t : \mathit{TypeBF}; \mathit{lr} : \mathit{ListResult}; \\
 & & \mathit{fbe} : \mathit{ForBoolCond} \\
\mathit{FixBoolFor} & \triangleq & \mathit{fixEle} : \mathit{varName}; \\
 & & \mathit{bf} : \mathit{BoolFor} \\
\mathit{TypeBF} & \triangleq & \mathit{and} \mid \mathit{or} \\
\mathit{ListResult} & \triangleq & \mathit{varName}^{+}
\end{array}
$$

As for a condition also for loops it is possible to assume conditional or normal forms. The expression executed inside a loop is similar to the condition expression but with some variations. The *ForMaCond* is a arithmetical expression that can use constants and variables, the difference with an *MaCond* is that it is not possible to use loops or functions inside this type of expression.

$$
\begin{array}{rcl}
\mathit{ForMaCond} & \triangleq & \mathit{ForMaITE} \mid \\
 & & \mathit{ForMaExp} \mid \\
 & & \mathit{constant} \mid \mathit{varName} \\
\mathit{ForMaITE} & \triangleq & \mathit{if} : \mathit{ForBoolCond}; \\
 & & \mathit{then}, \mathit{else} : \mathit{ForMaCond} \\
\mathit{ForMaExp} & \triangleq & \mathit{ForMaUnExp} \mid \\
 & & \mathit{ForMaBinExp} \\
\mathit{ForMaUnExp} & \triangleq & s : \mathit{sub}; \\
 & & \mathit{me} : \mathit{ForMaCond} \\
\mathit{ForMaBinExp} & \triangleq & \mathit{me}_1, \mathit{me}_2 : \mathit{ForMaCond}; \\
 & & \mathit{mo} : \mathit{MaOp}
\end{array}
$$

The *ForBoolCond* is a boolean expression and does not allow the use of loops or functions as the *ForMaExp*. Clarified this two points, all the elements the name of which

starts with *For* are equals to the elements used by an *BoolCond*.

$$
\begin{array}{rcl}
\mathit{ForBoolCond} & \triangleq & \mathit{ForBoolITE} \mid \\
 & & \mathit{ForBoolExp} \mid \mathit{ForComp} \mid \\
 & & \mathit{varName} \mid \mathit{constant} \\
\mathit{ForBoolITE} & \triangleq & \mathit{if}, \mathit{then}, \mathit{else} : \mathit{ForBoolExp} \\
\mathit{ForBoolExp} & \triangleq & \mathit{ForBoolUnExp} \mid \\
 & & \mathit{ForBoolBinExp} \\
\mathit{ForBoolUnExp} & \triangleq & n : \mathit{neg}; e : \mathit{ForBoolCond} \\
\mathit{ForBoolBinExp} & \triangleq & e_1, e_2 : \mathit{ForBoolCond}; \\
 & & \mathit{bo} : \mathit{BoolOp} \\
\mathit{ForComp} & \triangleq & \mathit{ce_1}, \mathit{ce_2} : \mathit{ForCompElem}; \\
 & & \mathit{co} : \mathit{CompOp} \\
\mathit{ForCompElem} & \triangleq & \mathit{ForMaCond} \mid \mathit{varName} \mid \\
 & & \mathit{constant}
\end{array}
$$

6. Risk Definition for the Running Example

We now have all ingredients to show how the risks that we identified for the Payment subprocess can be captured via sensor conditions, using the language defined in Section 5.

There is an overtime process if an instance of the payment subprocess is not completed within an MCT of five days (see Section 2). Accordingly, the corresponding risk can be detected if the probability of exceeding MCT is greater than a tolerance value (e.g. 60%). This condition is checked by using two variables: one to retrieve the amount of time T_c expired and the other to retrieve the T_e remaining cycle time.

d	=	5
T_c	=	case(current).Payment(PassTimeInMillis)
T_e	=	case(current).(TimeEstimationInMillis)

The risk condition defined to monitor this risk is:

$(1-(d*24*60*60*1000)/(T_e+T_c))>0.6$.

There is an approval fraud whenever a Senior Finance Officer approves two orders for the same customer within five days (see Section 2). Accordingly, the corresponding risk can be detected if given an order o of customer c to be approved, either of the following conditions holds: i) o has been allocated to a Senior Finance Officer who has already approved another order for the same customer in the last five days; or ii) at least one Senior Finance Officer is available who approved an order for customer c in the last five days and all other Senior Finance Officers who never approved an order for c during the five days are available.

This risk condition is triggered by an event, i.e. the spawning of a new instance of task Approve Shipment Payment Order (see Figure 1). This is checked by using a variable to retrieve the status of this task in the current instance. The risk condition itself is given by the disjunction of the two conditions described above. The first such condition is checked by using a variable to retrieve which resources were allocated to task Approve Shipment Payment Order, and another variable to retrieve the number of times this task was completed for customer c. This latter variable is defined via a case condition over customer c, the completion time of this task (that must be greater than the start time of the current task Approve Shipment Payment Order minus five days in milliseconds), and the identifier of the instance (that must be different from the identifier of the current instance).

The second condition is checked by using two variables and invoking two functions. A variable to retrieve which resources completed task Approve Shipment Payment Order, and another variable to retrieve all resources that can be offered this task (i.e. the current task). The first variable is defined

via a case condition over customer c and the completion time of this task (that must be greater than the start time of the current task Approve Shipment Payment Order minus five days). The two invoked functions return the number of tasks started on the resources that completed task Approve Shipment Payment Order, and the number of tasks in the execution queue of the resources that have been offered this task, and did not complete it for customer c in the last five days.

After the definition of the variables, defined in Table 1, the risk condition is specified as follows:

$(\mathsf{ASPO}_{\#\mathrm{App}}>0)\vee((\mathsf{sfo}_2.\mathsf{startMinNumber}=0)\wedge(\mathsf{sfo}.\mathsf{startMinNumberExcept}.\mathsf{sfo}_2>=1))$.

An order unfulfillment occurs whenever an order is updated more than five times (see Section 2). Accordingly, the respective risk can be detected if: i) task Update Shipment Payment Order is currently being performed for a given order; and ii) it is likely that this order will be updated again (i.e. task Update Shipment Payment Order will be repeated within the same process instance). The probability that the number of times a task will be repeated within the same instance of the Payment subprocess is computed by dividing the number of instances, where five executions for task Update Shipment Payment Order has been reached, over the number of instances that have executed this task at least as many times as it has been executed by the current instance, and have completed. The tolerance value indicates a threshold above which the risk should be notified to the user. For example, this threshold is 60%. This condition can be checked by using two variables: one to retrieve the amount of orders that have been updated at least as much as this order, the other to retrieve the amount of orders that have been updated at least five times.

The variables described can be defined via the sensor definition language as in Table 2. The risk condition is specified as follows:

$(\mathsf{USPO\#U5}/\mathsf{USPO\#US})>0.6$.

An underpayment fraud occurs whenever a customer underpays more than three times in a five-day time frame (see Section 2). Accordingly, the respective risk can be detected if i) task Issue Debit Adjustment is being performed for a given customer and order (this is the trigger for this risk); and ii) the probability that the maximum number of allowable executions for this task will be reached in a five-day time frame, is above the fixed tolerance value for this risk, say 60% (this is the risk condition itself). This condition can be checked by using two variables: one to retrieve the number of times the task Issue Debit Adjustment has been completed for this customer, the other to retrieve the probability that an attempted fraud will take place. For this second variable, we use the *Action* "`FraudProbabilityFunc`" to compute the specific probability (see Appendix A).

The defined variables are implemented through the sensor language as follows as in Table 3. These variables are used to compose the following risk condition:

$\mathsf{Probability}>0.6$.

The definition of actions and functions used in the above variables is provided in appendix. In particular, the complete list of all actions is provided in Appendix A, the complete lists of the nested loops and of

sfo_1	=	case(current).Approve_Shipment_Payment_Order_593(allocateResource)
c	=	case(current).Issue_Shipment_Invoice_594.ShipmentInvoice.Company
d	=	5
ASPO	=	case(current).Approve_Shipment_Payment_Order_593(OfferTimeInMillis)
ASPO#	=	case(Approve_Shipment_Payment_Order_593(completeResource)=sfo_1 ∧ Issue_Shipment_Invoice_594.ShipmentInvoice.Company=c ∧ Approve_Shipment_Payment_Order_593(CompleteTimeInMillis)> (ASPO-(d*24*60*60*1000)) ∧ (ID)!=[IDCurr]) .Approve_Shipment_Payment_Order_593(CountElements)
sfo_2	=	case(Issue_Shipment_Invoice_594.ShipmentInvoice.Company=c ∧ Approve_Shipment_Payment_Order_593(isCompleted)="true" ∧ Approve_Shipment_Payment_Order_593(CompleteTimeInMillis)> ($ASPO_{StartTime}$-(d*24*60*60*1000)) ∧ (ID)!=[IDCurr]) .Approve_Shipment_Payment_Order_593(completeResource)
sfo	=	case(current).Approve_Shipment_Payment_Order_593(offerDistribution)

Table 1: Variable definition for approval fraud risk.

USPO#UC	=	case(current).Update_Shipment_Payment_Order_604(Count)
USPO#U5	=	case(Update_Shipment_Payment_Order_604(Count)>=5). Update_Shipment_Payment_Order_604(CountElements)
USPO#US	=	case(Update_Shipment_Payment_Order_604(Count)>=USPO#UC ∧ Process_Shipment_Payment_603(isOffered)="true"). Update_Shipment_Payment_Order_604(CountElements)

Table 2: Variable definition for order unfulfillment risk.

the functions are provided in Appendix B and Appendix C.

7. Risk Templates

To facilitate the definition of concrete risks on process models, we defined a notion of risk template. A risk template is a risk defined using generic tasks and variables not associated with any real process model. In essence, a risk template is an "abstract" risk whose condition is composed of generic tasks and of generic variables associated with these tasks. These generic tasks and variables will then be bound to real tasks and variables when the template risk is used to define a concrete risk for a specific process model.

Several studies address different types of risks [57, 18, 46, 61, 66, 68]. After an analysis of these studies we decided to subdivide our risk templates into categories reflecting these types of risks. We expect that this division into categories also reduces the effort required by a user to select a suitable template. These categories were defined using the risk taxonomy proposed by Rosemann and zur Muehlen [57] and other types of risks proposed in the literature [27].

In particular we organized risk templates in two levels. The first level is obtained using the risk taxonomy, obtaining five categories of risks. These categories are: i) *Goal*, capturing risks of not achieving the process objectives; ii) *Structural*, capturing risks deriving from wrong decisions taken at design time; iii) *Data*, capturing risks of damaging data integrity; iv) *Technology*, capturing risks of impacting on the availability of IT systems; and v) *Organizational*, capturing risks of process faults that may impact

$\text{IDA}_{\text{StartTime}}$	=	case(current).Issue_Debit_Adjustment_605(StartTimeInMillis)
c	=	case(current).Issue_Shipment_Invoice_594.ShipmentInvoice.Company
d	=	5
$\text{IDA}_{\#\text{Issue}}$	=	case(Issue_Shipment_Invoice_594.ShipmentInvoice.Company=c ∧ Issue_Debit_Adjustment_605(Count)>0 ∧ Issue_Debit_Adjustment_605 (CompleteTimeInMillis)>($\text{IDA}_{\text{StartTime}}$-d*24*60*60*1000)) .Issue_Debit_Adjustment_605(CountElements)
GroupingElement	=	Issue_Shipment_Invoice_594.ShipmentInvoice.Company
WindowElement	=	Issue_Debit_Adjustment_605(CompleteTimeInMillis)
Threshold	=	0.6
Probability	=	case(Issue_Debit_Adjustment_605(Count)>0 ∧ (ID)!=[IDcurr]). Issue_Debit_Adjustment_605(FraudProbabilityFunc, $\text{IDA}_{\#\text{Issue}}$, 3, GroupingElement, WindowElement, (d*24*60*60*1000))

Table 3: Variable definition for underpayment fraud risk.

on the employees or be caused by them. For each of these risk categories we defined eleven subcategories. These subcategories are based on the type of risks proposed in the literature [27]. Specifically, we have: i) *Strategic*, risks of affecting the implementation of business strategies; ii) *Operations*, risks of affecting the capability of supplying and producing services or goods; iii) *Supply*, risks that prevent a resource from executing an operation; iv) *Customer*, risks related to customers; v) *Asset*, risks deriving from the use of an asset; vi) *Competitive*, risks deriving from faults related to competitiveness; vii) *Reputation*, risks of loss of reputation; viii) *Financial*, risks of financial loss; ix) *Fiscal*, risks caused by changes in taxation; x) *Regulatory*, risks related to changes in regulations; xi) *Legal*, risks of faults that may lead to legal actions. Figure 6 shows a mind map illustrating how risk templates are categorized and subcategorized.

We defined 14 risk templates as a starting point for a larger risk template repository. For example, one template of type Data/Operations, aims to detect a possible inconsistency in the value of a critical variable. The template requires a as the critical variable, b the identifier variable, and two set of instances s_1 and s_2, where s_1 is composed of the instances in which the values of a and b are equal to the value of a and b for the current instance, and s_2 is composed of the instances in which the value of a is equals to the value of a for the current instance. If the number of instances in s_1 divided by the number of instances in s_2 is greater than a certain threshold the template triggers a notification. This risk template can be used to represent a case in which the critical variable is a credit card number, or a bank account, while the identifier variable is the customer identifier.

Another risk template, of type Goal/Financial, detects the possibility that a particular process may exceed the budget assigned for its execution. This risk is detected looking at all the completed instances of the process and at all the completed instances with an execution cost at least equal to the execution cost of the current instance. The condition calculates the probability that the current instance will exceed the budget. This risk can be associated with any process where the cost is a relevant element. An example is an insurance process where opinions from different assessors may be required but the

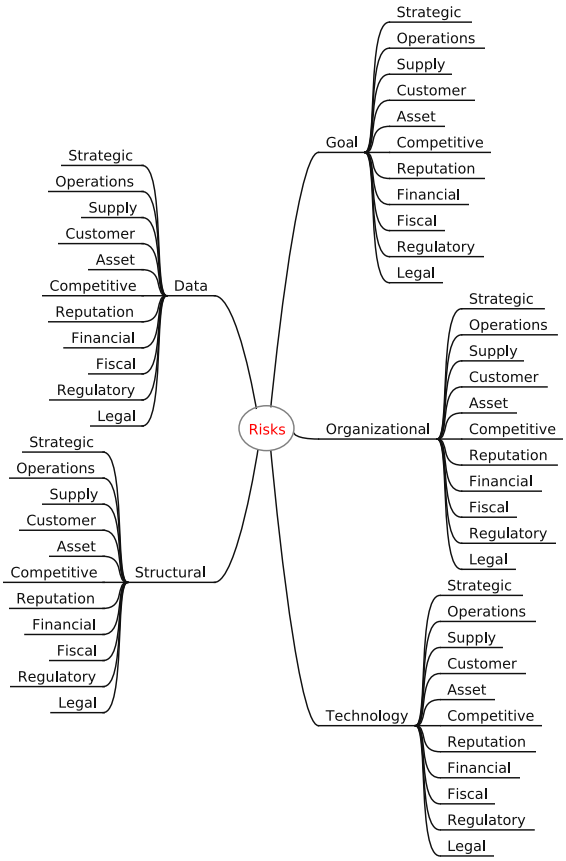


Figure 6: Template structure with categories and subcategories.

total cost of their involvement should not exceed the premium.

In the category Organizational, subcategory Supply, we defined a template that addresses possible delays with the execution of tasks. It detects the possibility that a critical task would not be started as soon as it is offered. This risk is detected looking at all the resources which have been offered a work item of the critical task. If the ratio between the number of resources that are busy versus the total number of resources is greater than a certain threshold the risk is detected. This type of risk is relevant for processes in healthcare such as a process for transferring organs from one hospital to another, where the unavailability of a resource may cause the organ deterioration. Finally, taking in consideration the process described in Section 2, let's say that the company wants to avoid that processing a shipment payment may be subjected to delays, what we have to do is: i) import the template; ii) create a mapping of our generic critical task to the task "Process Shipment Payment"; and iii) modify the specified threshold if required.

These templates are available as part of our implementation in the YAWL system (see Section 8). Finally, a briefly overview of the templates available at the moment is provided in Table 4. In addition to the three templates showed before we have templates that identify risk of: underpayment fraud, approval fraud, time limits exceeding, four-eyes principle violations and others.

8. Software Implementation

In order to prove the feasibility of our approach, we implemented the sensor-based architecture in the YAWL system.[2] We decided to extend the YAWL system for the following reasons. First, this system is based on a service-oriented architecture, which facilitates the seamless addition of new services. Second, the system is open-source, which facilitates its distribution among academics and practitioners, and widely used in practice (the system has been downloaded over 100,000 times since its first inception in the open-source community). Finally, the underlying YAWL language is very expressive as it provides wide support for the workflow patterns [74].

As part of this implementation, we extended the YAWL Editor version 2.2beta

[2]Available at `www.yawlfoundation.org`

Category	SubCategory	Template Name	Risk Description
Data	Financial	Underpayment Fraud	Underpayment fraud
	Operational	Data Inconsistency Across Cases	Wrong information provided, detected using multiple cases
		Data Inconsistency Parallel Activity	Wrong information provided to parallel tasks
Goal	Financial	CostProcessExceededRisk	Exceeding the budget during the execution of the process
	Strategic	Activity Time Exceeded	Exceeding the time limit within which the activity needs to be completed
		MultiActivity Time Exceeded	Exceeding the time limit within which a sequence of activities needs to be completed
		Process Time Exceeded	Exceeding the time limit within which the process needs to be completed
Organizational	Financial	Approval Fraud	Approval fraud
	Operational	Four-Eyes Principle	Four-Eyes Principle violation
		Four-Eyes Principle Across Cases	Four-Eyes Principle violation for activities across different cases
		Inadequate Preparation	The activity is executed by a novice resource
	Supply	Delay In Start	The activity start will be delayed
		Task Priority Unfulfilled	Activity with high priority cannot start because resources are busy
Structural	Operational	Loop	A loop is executed too many times

Table 4: Risk Templates and descriptions

with a new component, namely the Sensor Editor, for the specification of sensors within YAWL process models. Such graphical component, shown in Figure 7, fully supports the specification of sensor conditions as defined in Section 4, and the specification of risk templates.

A wizard (see figure 8) was developed as part of the YAWL Editor to facilitate the use of risk templates and in particular the mapping required for their use. This wizard using an user friendly interface guides the user during each step of the mapping, providing a description of each generic task and variable. The wizard also provides the possibility of customizing a risk, since it is possible to modify a risk condition during the creation of a risk using templates.

Moreover, we implemented the Sensor Manager as a generic component which exposes three interfaces (engine, database and monitor) as described in Section 4. We then wrapped this component into a Web service which implements the three interfaces for the YAWL system, allowing the component to interact with the YAWL Engine, the Monitor service and the YAWL database. While there is a straightforward mapping between the YAWL Engine and our engine interface, and between the YAWL Monitor service and our monitor interface, we had to join several YAWL tables to implement our database interface. This is because in the YAWL system, event logs are scattered across different database tables. For example, to retrieve all identifiers of the process instances for a specific process model, given the model identifier, we need to perform a join among the following YAWL tables: `logspecification`, `lognetinstance`, `lognet` and `logevent`.

The complete mapping is illustrated in Table 5. As an example, this table also shows the mapping between our database interface and the relational schema used by Oracle BPEL 10g to store BPEL process logs. Also in this case, the database can be fully mapped by joining several tables.

Finally, we implemented a separate service to estimate the remaining cycle time

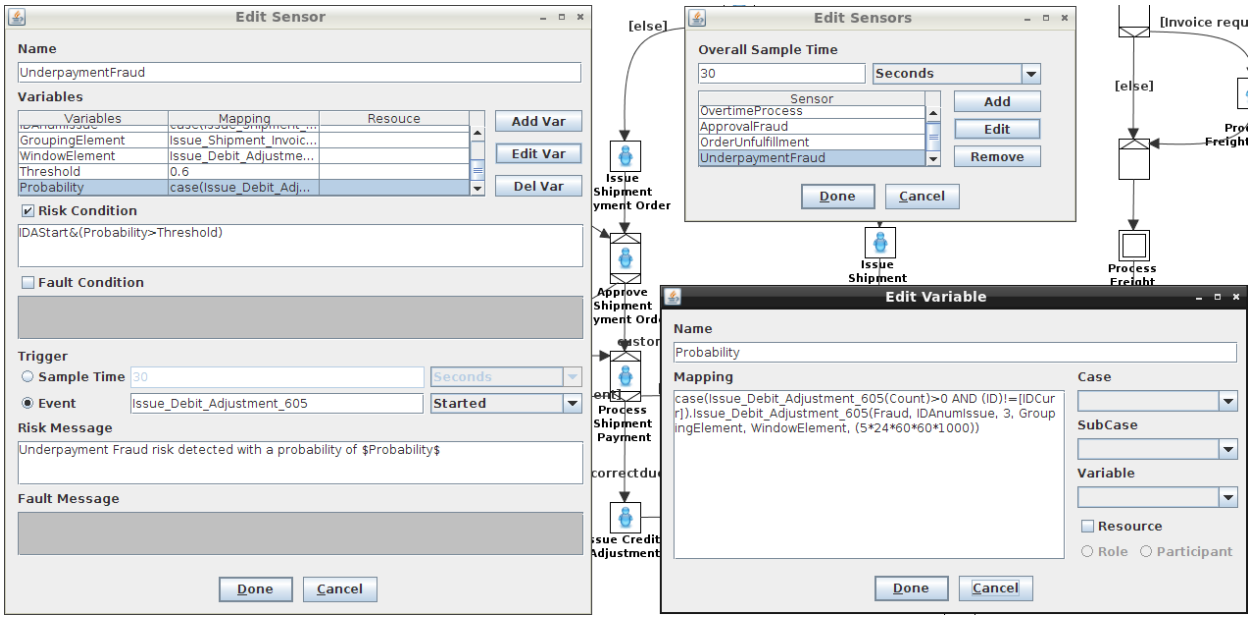


Figure 7: The Sensor Editor within the YAWL Editor.

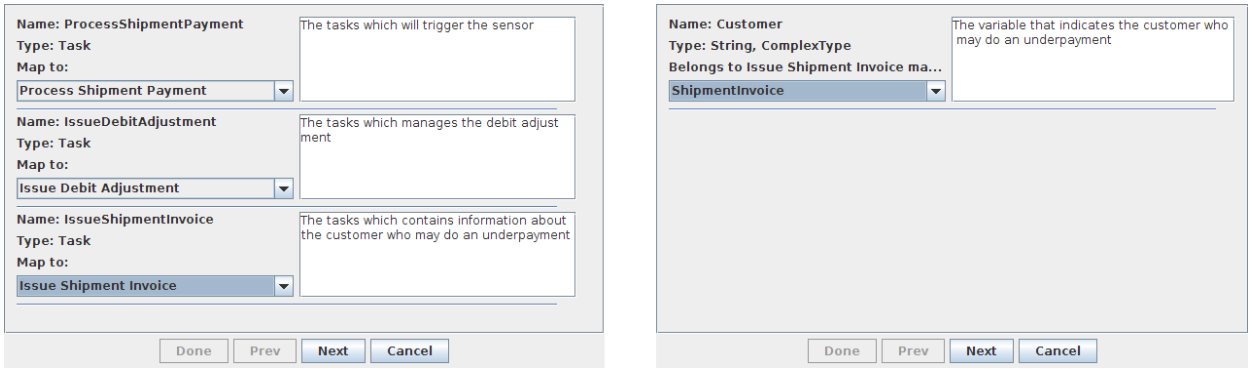


Figure 8: Template Wizard: a) Tasks mapping, b) Variables mapping.

T_e for a process or task instance. This service uses ProM's prediction miner [76] to compute the estimations, and provides the results to the Sensor Manager on demand. While the estimation of T_e could be done on-line, i.e. while evaluating a particular sensor condition at run-time, parsing the full logset each time would be inefficient. Rather, we compute this estimation off-line, whenever a new process model is deployed to the YAWL Engine, by using the logset available at that time. Periodically, we update the logset with the new instances being executed meantime, and invoke this service to refresh the estimations for each process model currently deployed.

9. Evaluation

In this section we discuss two evaluations of the sensor-based architecture. First, we discuss a performance analysis of the implementation of the architecture in the YAWL system, and second, we report on a usabil-

Database table	Tables that need to be joined	
	YAWL	**Oracle BPEL 10g**
WorkFlowDefinition	logspecification, lognet, lognetinstance, logevent	cube_instance and cube_scope
SubProcess	logspecification, lognet, lognetinstance, logevent	cube_instance and cube_scope
Activity	lognetinstance, logtask, logtaskinstance, lognet, logevent, logspecification, rs_eventlog	wftask and work_item
Variables	logtask, lognet, lognetinstance, logtaskinstance, logevent, logdataitem, logspecification	audit_trail, audit_detail and xml_document
Role	rs_participant	wftask
ActivityRole	rs_eventlog, logtaskinstance	wftask

Table 5: Database interface mapping for YAWL 2.2beta and Oracle BPEL 10g.

ity evaluation of the architecture on basis of a survey with 21 BPM practitioners that gained experience with the system.

9.1. Performance Analysis

We used our implementation to evaluate the scalability of the approach. First, we measured the time needed to evaluate the basic functions (e.g. counting the number of instances of a task or retrieving the resource allocated to a task). Next, we measured the time needed to evaluate the sensor conditions for the risks defined in the Payment subprocess. The tests were run on an Intel Core I5 M560 2.67GHz processor with 4GB RAM running Linux Ubuntu 11.4. The YAWL logs were stored on the PostGres 9.0 DBMS. These logs contained 318 completed process instances from 36 difference process models, accounting for a total of 9,399 process events (e.g. task instance started and completed, variable's value change). Specifically, there were 100 instances from the Payment subprocess yielding a total of 5,904 process events. The results were averaged over 10 runs.

Table 6 shows the results of the evaluation of the basic functions provided by our language. In particular, in this table we compare the evaluation times obtained by accessing the YAWL logs via our database interface, with those obtained by accessing a serialization of the logs, e.g. in the OpenXES format. While OpenXES provides a simple and unique representation of a generic set of process logs, accessing an OpenXES file in real-time, i.e. during the execution of a process instance, is not feasible, due to the long access times (e.g. 6.5 sec. on average for evaluating a net variable). On the other hand, accessing the logs via our database interface, despite it requires the creation of a specific implementation for each BPMS database, provides considerably faster times than accessing OpenXES files (at least 87% gain w.r.t. OpenXES access). In fact, as we can see from Table 6, the evaluation times for all the basic functions are below 30 ms, apart from function `task variable`, which takes almost 100 ms and function `net variable`, which takes about 430 ms.

The last two basic functions reported in Table 6, namely `task distribution` and `task initiator`, are evaluated in less than 250 milliseconds. These functions are not computed by accessing the logs, but rather by accessing information that is contained directly in an executable process model, e.g. the resources that are associated with a specific task. However, in our implementation we still use the database interface to access this information, in order to provide the developer with a single access point to all process-related data.

Table 7 reports the results of the eval-

Basic function	Description	OpenXES time [ms]	Database time [ms]	Reduction rate [%]
net status	functions checking if a net status has been reached (isStarted, isCompleted)	6,535	18.9	99.71
net time	functions returning the time when a net status has been reached (startTime, completeTime, startTimeInMillis, completeTimeInMillis)	6,781	18.8	99.72
net variable	returns the value of a net variable	6,489	432.6	93.33
task count	number of times a task has been completed	803	19.8	97.53
task resource	functions that return the resources associated with a task (offerResource, allocateResource, startResource, completeResource)	850	20.9	97.54
task status	functions checking if a task status has been reached (isOffered, isAllocated, isStarted, isCompleted)	792	30.5	96.14
task time	functions returning the time when a task status has been reached (offerTime, allocateTime, startTime, completeTime, offerTimeInMillis, allocateTimeInMillis, startTimeInMillis, completeTimeInMillis)	824	22.3	97.29
task variable	returns the value of a task variable	787	96.7	87.71
task distribution	functions returning the resources associated with a task by default (offerDistribution, allocateDistribution, startDistribution, completeDistribution)	243		-
task initiator	functions returning the allocation strategy for a resource association (offerInitiator, allocateInitiator, startInitiator, completeInitiator)	249.6		-

Table 6: Performance of basic functions.

uation of the sensor conditions defined for our running example. While the sensor conditions for the overtime process and order unfulfillment faults are very low (below 150 ms), longer times are obtained for evaluating the conditions for the two faults related to fraud. This is because both these conditions require to evaluate "complex queries", i.e. queries over the entire process logs: in the approval fraud, we need to retrieve all resources that approved an order for a specific customer, while in the underpayment fraud we need to retrieve all process instances where a debit adjustment was issued and aggregate these instances per customer. These queries are different than those needed to evaluate the basic functions, as the latter are performed on the events in the logs that are relative to a single known process instance, e.g. the instance for which the sensor condition is being evaluated.

The worst-case complexity of evaluating

Sensor	Min [ms]	Max [ms]	Ave [ms]	St.Dev.
Overtime process	121	137	131.8	4.66
Approval fraud	6,483	7,036	6,766.4	183.06
Order unfulfillment	69	91	77.4	7.18
Underpayment fraud	3,385	3,678	3,523	89.98

Table 7: Performance of sensors.

one such a complex query is still linear on the number of parameters that need be evaluated in the query (corresponding to the language element *CondExprSet* in Section 4) multiplied by the total number of instances present in the logs (corresponding to the size of table WorkflowDefinition addressed by our database interface).

In conclusion, the performances of evaluating sensor conditions should always be considered w.r.t. the specific process for which the risks are defined, and the type of trigger used. For example, let us assume an average duration of 24 hours for the Payment subprocess, with a new task being ex-

ecuted every 30 minutes. This means we have up to 30 minutes to detect an overtime process risk before a new task is executed, and we need to compute this sensor condition again. If we choose a rate of 5 minutes to sample this condition, we are well below the 6 minute-threshold, so we can check this sensor condition up to 6 times during the execution of a task. Since we do this in less than 150 ms, this time is acceptable. For an event-driven risk we also need to consider the frequency of the specific event used as trigger. For example, the approval fraud risk is triggered every time an instance of task Approve Shipment Payment Order is offered to a Senior Financial Officer for execution. Since we take up to 7 seconds to compute this sensor condition, we are able to cope with a system where there is a request for approval every 7 seconds. So also for this sensor, the performance is quite acceptable.

9.2. Usability Analysis

For the evaluation of the usability of the sensor-based architecture we followed the example from related studies [39, 40] and conducted focus group-like online sessions with BPM practitioners. Each online session consisted of a tutorial about the functions and use of the sensor-based architecture that consisted of an introduction, the creation of risk sensors in the YAWL editor and detailed instructions for defining sensors from scratch or on the basis of templates. Additional documentation provided descriptions and explanations of actions, resource functions and loops relevant to risk sensors. After completing the tutorial, each participant was asked to complete a structured online questionnaire (see Appendix D).

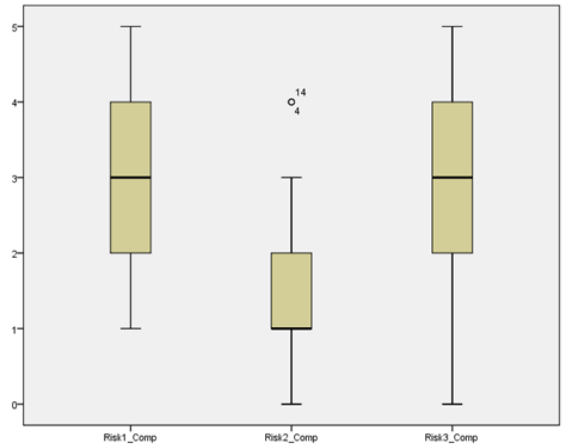
In these focus groups, overall twenty-one participants participated, where 10 of them were from Italy and 11 from Australia. Participants, on average, had about 2.6 years of experience with process modeling and had read and created, on average, 71 and 15 models, respectively, over the last twelve months. Participants' experience with different process modeling languages varied, with most having experience, at least, with UML Activity Diagrams, Petri Nets, EPCs and/or BPMN. These characteristics describe our participants as proxies for novice to average BPM professionals, with one of our participants being representative of an expert practitioner (more than 5 years experience, more than 250 models created, more than 1000 models read). Overall, our study population might not be representative of experts but is roughly equivalent to the range of expertise found in actual BPM practitioners as reported in other surveys, e.g. [55].

The structured questionnaire that was completed by the participants contained measurement items that were, where possible, adapted from prior surveys on process modeling [39, 40] and usage of the YAWL system [56]. The questionnaire was structured in four parts. First we gathered demographic information about the participants. Next, we captured information about the extent and intensity of usage experiences with YAWL or other BPMSs. Third, we presented three risk scenarios on the basis of the developed sensor-based architecture and examined how well participants were able to assess risks, in terms of accuracy of understanding the risks and the difficulty of understanding the meaning of the risks. Accuracy and difficulty are widely used measures [11, 44] of the effectiveness and the efficiency of the sensor-based approach in

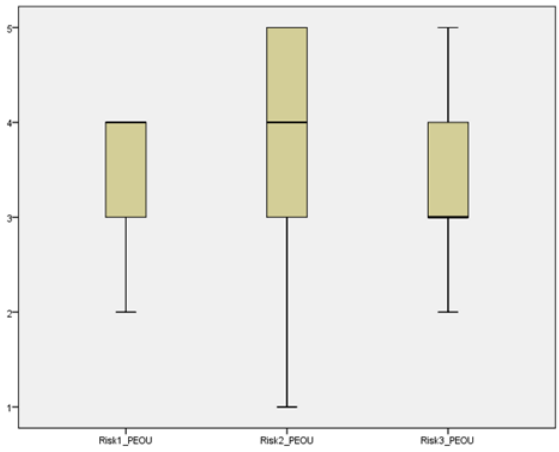
terms of how well users of the approach can understand the risk information provided through the approach, and how cognitive effort is required to develop this understanding. Three different scenarios were proposed to the users. The first and last scenarios describe risks using the developed risk definition language. Here we asked questions about the risk proposed. The second scenario describes a risk using a fault tree. Here we asked questions about how such a risk can be defined using the proposed language. For each scenario, a 5-point scale measured the difficulty of understanding the risk from very simple to very difficult, and five *true/false/do not know* questions measured accuracy of comprehension of the risk information provided.

Fourth and finally, we captured participants' perceptions about the usage experience of the approach, building on theories of technology acceptance [19] and continued usage of process modeling [53, 54]. To that end, we adapted the measurements items in [56] to measure satisfaction (SAT), usefulness (PU), ease of use (PEOU) and intentions to use (ITU). Similar to previous uses of these items [53, 54, 56], the measures were of appropriate reliability, with Cronbach's Alphas ranging from 0.64 (ITU) to 0.81 (PU).

The box-plots in Figure 9(a) and Figure 9(b) show the accuracy of the comprehension of the risks in the three scenarios as well as the perceived difficulty of understanding these risks. The data shows that scenarios 1 and 3 were reasonably well understood (averages for the comprehension questions were 2.9 for scenario 1 and 3.1 for scenario 3, both exceeding 50% accuracy), while comprehension of scenario 2 was significantly lower (average score 1.7). Similarly, perceived difficulty was highest for scenario 2 (average of 3.7), while the perceived difficulty for scenario 1 and 3 was similar in range (averages 3.4 and 3.3, respectively).



(a) Accuracy of risk comprehension, on a scale from 0 (low) to 5 (high).



(b) Difficulty of risk comprehension, on a scale from 1 (very simple) to 5 (very difficult).

Figure 9: Accuracy and difficulty of risk comprehension.

Next we were interested in understanding under which circumstances participants were able to obtain higher levels of accuracy in understanding the risk information provided. To that, we built a regression model in which we regressed relevant demographic data and perceived difficulty onto the to-

tal comprehension score across all three risk scenarios. Specifically, we included as coefficients:

- PMExp(Years): Process modeling experience in years
- PMExp(Training): Extent of formal training in process modeling in days over the last twelve months
- PMExp(SelfEducation): Extent of self-education in process modeling in days over the last twelve months
- PMExp(processModelsCreated): Number of process models created over the last twelve months
- BPMSFAM: Self-perceived familiarity with BPMSs [53]
- YAWLUse(time): length of use of the YAWL system
- YAWLUse(features): Number of YAWL features used
- YAWLUse(models): Number of YAWL models created, read or edited over the last twelve months
- RISKPerDif: Average perceived comprehension difficulty across the three presented scenarios.

The regression model was significant ($F = 3.54$, $p = 0.04$) and explained 77.9 percent of the variance in total comprehension score, thus attesting to very good explanatory power. Table 8 gives the results from the regression model analysis.

Table 8 shows that four factors were significant predictors for explaining comprehension accuracy, these being process modeling experience in years, number of process models created, use of the YAWL system and number of YAWL models created, edited or read over the last twelve months. Several interesting findings are noteworthy. First, PMExp(Years) and YAWLUse(models) are negative predictors, which may suggest that novice users with little process modeling experience and little experience with YAWL models benefited more from the risk sensor approach. Second, the difficulty of understanding the risk information provided is not related to how well users understand the risks. Third, training in process modeling or more varied use of the YAWL system, likewise, are irrelevant in terms of understanding risk information provided by the sensor-based architecture.

Variable	Standardized Coefficients		
	Beta	t	Sig.
PMExp(Years)	**-0.80**	**-2.79**	**0.02**
PMExp(Training)	-0.03	-0.16	0.88
PMExp (processModelsCreated)	**1.28**	**4.31**	**0.00**
BPMSFAM	0.38	1.15	0.28
YAWLUse(time)	**0.80**	**3.07**	**0.01**
YAWLUse(features)	-0.30	-1.40	0.20
YAWLUse(models)	**-1.20**	**-3.86**	**0.00**
RISKPerDif	-0.19	-1.05	0.32

Table 8: Regression analysis of risk comprehension across all three scenarios

Finally, we were interested in the participants' evaluation of the sensor-based architecture. Following the theory of technology acceptance [19] and its extended application in the process modeling context [54], we understand that satisfaction, ease of use and perceived usefulness are key criteria for explaining intentions to use a process modeling artifact. Figure 10 shows the box-plots for the average total factor scores for these four criteria.

The data displayed in Figure 10 suggests that participants rated the usefulness of the

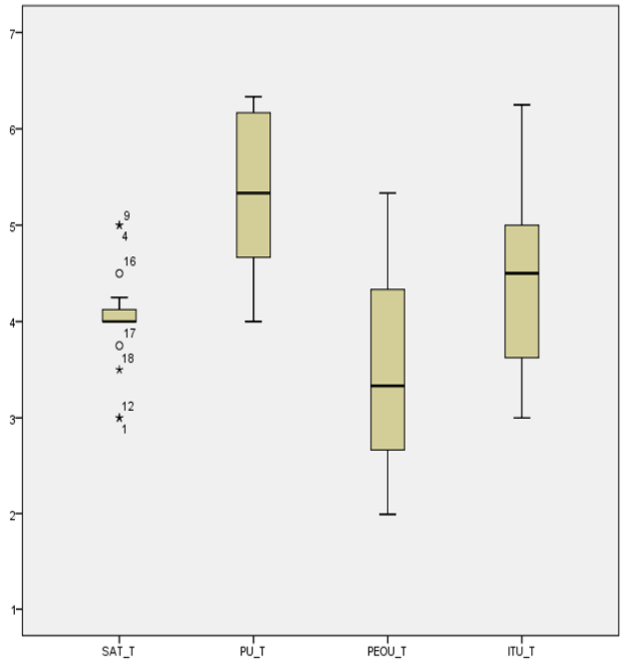


Figure 10: Perceptual evaluations of the sensor-based architecture on a scale from 1 (low) to 7 (high).

approach high (mean score > 5.3) and were in general inclined to use the system (mean score > 4.4). Satisfaction with the approach was reported as average (mean score $= 4.0$) and ease of use, notably, was reported as low (mean score < 3.5), indicating potential to improve interface and user interaction of the system. In a post-hoc analysis, we compared evaluations of SAT, PU, PEOU and ITU across users that scored low or high on risk comprehension, based on a median split, and across users that rated the perceived difficulty of understanding risks as low or high, again based on a median split. MANOVA tests in both cases showed that evaluations of the approach were independent from comprehension accuracy or difficulty, with p-values ranging from 0.30 to 0.49 and 0.18 to 0.58, respectively. The results indicate that the evaluations of satisfaction, usefulness, ease of use and intentions to use were robust against variances in the performance in using the system.

In summary, our evaluation showed that the sensor-based risk identification and modeling approach provided value in allowing participants to develop an understanding of process risks. We found the approach to be useful particularly for users with limited experience in process modeling or BPMSs. The evaluation further revealed that ease of use of the system should be improved to warrant better user acceptance.

10. Related Work

Risk measurement and mitigation techniques have been widely explored in various fields. At the strategic-level risk management, standards prescribe generic procedures for identifying, analyzing, evaluating and treating risks (see e.g. [69]). Although helpful, such general guidelines are inevitably vague and fail to provide any specific guidance for operationalizing risk management strategies in business processes. At the other extreme, there are many techniques for identifying risks in specific areas such as employee [1], conflict of interest [42] and in the engineering field more generally [30, 10]. Other approaches, such as fault-tree analysis [12], are general enough to be applied to multiple domains. However, none of these approaches provides insights on how to define and operationalize the detection of process-related risks. In the following, we first discuss related work at methodological level with respect to the proposed approach and then related work with respect to the architectural level of our proposal is discussed.

Previous process-based research recognizes the importance of explicitly linking elements of risk to business process mod-

els, through specific methodological approaches. Such approaches, discussed below, can be mainly categorized into design time Risk-aware BPM with and without integrated risks constructs. The former [57, 80, 33, 75, 62, 65, 63, 64, 17, 16, 79, 3] analyzes and models BPM risks through the introduction of new integrated risk constructs whereas the latter [48, 49, 59, 58, 8, 29, 70, 38, 73, 52, 41, 9, 32, 4, 5, 22, 23, 36, 2, 7, 60, 34, 67, 37] reuses existing risk analysis methods. In the next paragraphs we are going to discuss the most relevant of these approaches while we refer to [71] for a comprehensive discussion on all these approaches.

Among these approaches we can identify several groups. The first group is composed of the approaches described in [17, 16, 79, 48, 49, 59, 58, 62, 65, 63, 64, 41]. They propose to extend existing modeling languages, such as Business Process Model and Notation (BPMN), Event-driven Process Chain (EPC), Value-Focused Process Engineering (VFPE), semantic business process modeling language (SBPML) and the integrated definition (IDEF) language, with risk-related constructs. The approaches provide the possibility to annotate business process models with risk-related information and in some cases it is also possible to annotate mitigation actions. Furthermore Rotaru et al. [49, 59, 58] also propose an utility calculation technique that can be used to determine optimal risk countermeasure solutions.

The second group is composed of approaches [2, 7, 60, 8, 4, 5] which propose risk-informed design. In these approaches common is the idea of modelling business processes (in one or several stages) which contain elements of risk and possible mitigation actions. These models are then used to create a process model in which mitigation activities are already part of the process itself. In particular, Betz et al. [8] propose a method based on simulation to choose the optimum process model variant if several variants are proposed.

The third group focuses on simulation and is composed of the approches described in [33, 75, 73, 36, 34]. The ROPE (Risk-Oriented Process Evaluation), methodology proposed by Jakoubi et al. [33, 75], is based on the observation that faults (here called "threats") impact the functionality of resources (required for the execution of process activities). If a threat is detected, a countermeasure process or a recovery process is invoked to counteract the threat. The ROPE methodology aims to incorporate these aspects in a single model that can be simulated to determine a companys critical business processes and single points of failure. Similarly, Taylor et al. [73], propose a simulation environment based on the jBPM Process Definition Language (JPDL) workflow language. In this environment a process models annotated with risk information (i.e. key risk indicator (KRI), key performance indicator (KPI), and risk event) can be simulated in order to evaluate the effects of risk events on some predefined KPIs and KRIs. Finally, Kaegi et al. [36] simulate a process model described in BPMN via agent-based modelling technique to analyze business process-related risks, while Jallow et al. [34], propose an approach where risks in business processes are analysed. The approach use the Monte Carlo simulation [45], on a given a set of identified risk events and their occurrence probabilities, in order to assess and quantify the impact/consequences of those risk events (in terms of time, cost, performance, and other objectives) on each process activity and on the overall process.

Other works which cannot be grouped together annoverate: i) a taxonomy of process-related risks [57, 80, 49], which includes five process-related risk types (goals, structure, information technology, data and organization) that can be captured by four interrelated model types (risk structure model, risk/goal matrix, risk state model, and an extension to the EPC notation); ii) an extended goal-risk framework [3], which consists of an asset layer (composed of business process goals, activities, and business artifacts), an event layer (composed of various events, including risk events, that can impact the asset layer), and a treatment layer (composed of a set of risk treatment activities that can mitigate the impact of the risk events modeled in the event layer); and iii) a technique to evaluate a workflow's non-completion risk due to uncertain/dynamic information [67], which quantifies the confidence level of the non-monotonic predicates of a workflow and whether one of these confidence levels is below certain threshold considers the workflow to be risky suggesting the use of a backup workflow.

Finally, in the approach proposed by Kang et al. [37], a technique to estimate the probability that a process instance enters an abnormal termination state is defined. Process-related historical data is used to inform the probability estimation calculation. Then, a run-time risk estimation algorithm is developed such that appropriate risk alerts can be produced when risky situations are detected. The main limitation of this approach results be necessity of having an exhaustive log containing all possible executions and the impossibility of making distintions among different abnormal termination states.

With respect to the risk-aware BPM lifecycle shown in Figure 3, all the above proposals (except the approach by Kang et al. that is specifically focused on risk-aware workflow execution) only cover the phases of risk analysis and risk-aware process modeling (see Table 9). None of them specifies how risk conditions can be concretely linked to run-time aspects of process models such as resource allocation, data variables and control-flow conditions, for the sake of detecting risks during process execution. Thus, none of these approaches operationalizes risk detection into workflow management systems. Moreover, they neglect historical process data for risk estimation. As such, these approaches are complementary to our work, i.e. they can be used at a conceptual level for the identification of process-related risks, which can then be implemented via our sensor-based lower-level methodology.

From the architectural point of view, our approach, specifically our sensor-based architecture, is also related to real-time monitoring of business process execution. Similarly to our approach, Oracle Business Activity Monitoring (BAM) [51] relies on sensors to monitor the execution of BPEL processes. Three types of sensors can be defined: activity sensors, to grab timings and variable contents of a specific activity; variable sensors, to grab the content of the variables defined for whole BPEL process (e.g. the inputs to the process); and fault sensors, to monitor BPEL faults. These sensors can be triggered by a predefined set of events (e.g. task activation, task completion). For each sensor, one can specify the endpoints where the sensor will publish its data at run-time (e.g. a database or a JMSQueue). We allow the specification of more sophisticated sensor (and fault) conditions, where different process-related aspects can be in-

Approach	Risk identification/ Risk analysis	Process design/ Risk-aware process modeling	Process implementation/ Risk-aware workflow implementation	Process enactment/ Risk-aware workflow execution	Process diagnosis/ Risk monitoring & mitigation
zur Muehlen et al. [57, 80]	✓	✓			
Asnar and Giorgini [3]	✓	✓			
Karagiannis et al. [38]	✓	✓			
Singh et al. [67]	✓	✓			
Cope et al. [17, 16]	✓	✓			
Weiss and Winkelmann [79]	✓	✓			
Mock and Corvo [48]	✓	✓			
Rotaru et al. [49, 59, 58]	✓	✓			
Sienou et al. [62, 65, 63, 64]	✓	✓			
Lambert et al. [41]	✓	✓			
Jakoubi et al. [33, 75]	✓	✓			
Taylor et al. [73]	✓	✓			
Kaegi et al. [36]	✓	✓			
Jallow et al. [34]	✓	✓			
Bergholtz et al. [2, 7, 60]	✓	✓			
Betz et al. [8]	✓	✓			
Bhuiyan et al. [4, 5]	✓	✓			
Hermann and Hermann [29]	✓	✓			
Strecker et al. [70]	✓	✓			
Panayiotou et al. [52]	✓	✓			
Bhuiyan et al. [9, 32]	✓	✓			
Fenz et al. [22, 23]	✓	✓			
Kang et al. [37]				✓	
Our proposal	✓	✓	✓	✓	✓

Table 9: Comparison of available R-BPM approaches with respect to the five phases of our reference R-BPM lifecycle.

corporated such as data, resource allocation strategies, order dependencies, as well as historical data and information from other running process instances. Moreover, our sensors can be triggered by process events or sampled at a given rate. Nonetheless, our sensor-based architecture is exposed as a service and as such it could be integrated with other process monitoring systems, such as Oracle BAM.

Real-time monitoring of process models can also be achieved via Complex Event Processing (CEP) systems. In this context, CEP systems have been integrated into commercial BPMSs, e.g. webMethods Business Events[3], ARIS Process Event Monitor [20] and SAP Sybase [72], as well as explored in academia [24, 28]. A CEP system allows the analysis of aggregated events from different sources (e.g. databases, email accounts as well as process engines). Using predefined rules, generally defined with a specific SQLlike language [78], a CEP system can verify the presence of a specific pattern among a stream of simple events processed in a given time window. Our approach differs from CEP systems in the following aspects: i) strong business process orientation vs general purpose system; ii) ability to aggregate and analyze complex XML-based events (e.g. process variables) vs simple events; iii) time-driven and event-driven triggers vs event-driven trigger only. Moreover, CEP systems typically suffer from performance overheads [28, 78] which limit their applicability to real-time

[3]http://www.softwareag.com/au/products/wm/events/overview

risk detection [78].

This article is an extended version of the work presented in [14]. Compared to this work, this article provides the full and revised definition of the language's abstract syntax, the support for risk templates, the usability evaluation with users and a comprehensive related work.

11. Conclusion

We contributed an approach for real-time monitoring of risks in executable business process models. The approach embeds elements of risk into each phase of the BPM lifecycle: from process design, where high-level risks defined via a risk analysis method are mapped down to specific process model elements such as activities, resources and data, through to process diagnosis, where risks are detected during process execution, and those no longer tolerable are notified to process administrators. To the best of our knowledge, this is the first attempt to concretely embed risks into executable business processes and enable their automatic detection at run-time.

As a second contribution, we provided an operationalization of the proposed risk-awareness approach on top of BPMSs. This is achieved via a distributed, sensor-based architecture that communicates with a BPMS via a set of tool-independent interfaces. Each risk is associated with a sensor condition and refers to a fault, which is an undesired state of the process. Conditions can relate to any process aspect, such as control-flow dependencies, resource allocations, the content of data elements, both from the current process instance and from instances of any process that have already been completed. At design-time, these conditions are expressed within a process model via a simple query language, for which we provide an abstract syntax. At run-time, each sensor independently alerts a sensor manager when the associated risk condition evaluates to true during the execution of a specific process instance. When this occurs, the sensor manager notifies a process administrator about the given risk by interfacing with the monitoring service of the BPMS. This allows early risk detection which in turn enables proper remedial actions to be taken in order to avoid potentially costly process faults.

From an analysis of relevant literature, we designed a set of risk templates to allow process designers to easily specify new risk conditions into a process model. Each template captures an abstract risk. To use these templates, one has to bind the template variables to concrete elements of the process model for which the risk condition needs to be monitored. We contend that by using such templates the effort of defining risks in executable process models can be reduced.

As a proof-of-concept, we implemented the sensor-based architecture on top of the YAWL system along with 14 representative templates. We then used the tool to evaluate the feasibility of the approach in practice. This was carried out in two directions. First, we evaluated the performance of the implementation; second, we evaluated the usability and ease of use of the approach with users of the YAWL system. The performance experiments showed that the sensor conditions can be computed efficiently and that no performance overhead is induced to the BPMS engine. The results of the empirical evaluation with users showed that the sensor-based risk identification and modeling approach provided value to develop an understanding of process risks. We

found the approach to be useful particularly for users with limited experience in process modeling or BPMSs. The evaluation further revealed that ease of use of the language for defining sensors should be improved to warrant better user acceptance. In order to overcome this limitation we plan to devise a mechanism for automatically deriving skeletons of sensors directly from fault trees. We expect that this will allow a smoother transition from high-level risk definitions to low-level risk conditions. Moreover, aiding features provided by common source code editors such as autocomplete, syntax highlighting and bracket matching can be put in place to reduce the human effort.

The approach presented in this paper and its operationalization serve as the cornerstone for other techniques that aim to bridge the gap between risk and process management. In particular, in [15] we documented a technique which uses input from this approach in order to mitigate risks. Specifically, as soon as one or more risks are detected which are no longer tolerable, the technique identifies a set of alternative mitigation actions that can be applied by process administrators. A mitigation action is a sequence of controlled changes applied to the process instance, that takes into account a snapshot of the process resources and data, and the current status of the system in which the process is executed. Furthermore, in [13] we reported on a second technique which also builds on the approach presented in this paper to allow process participants to make risk-informed decisions. This is achieved by estimating the risk that the current process instance will end up with one or more faults based on the input provided by the participant, e.g. when filling out a user form based on the conditions specified for each fault.

Acknowledgments We thank Peter Hughes for his help with the identification of process-related risks. This research is partly funded by the ARC Discovery Project "Risk-aware Business Process Management" (DP110100091). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] Albrecht, W.S., Albrecht, C.C., Albrecht, C.O., Zimbelman, M.F., 2008. Fraud Examination. 3rd ed., South-Western Publishing.

[2] Andersson, B., Bergholtz, M., Edirisuriya, A., Ilayperuma, T., Johannesson, P., 2005. A declarative foundation of process models, in: Pastor, O., e Cunha, J.F. (Eds.), CAiSE, Springer. pp. 233–247.

[3] Asnar, Y., Giorgini, P., 2008. Analyzing business continuity through a multi-layers model, in: Dumas, M., Reichert, M., Shan, M.C. (Eds.), BPM, Springer. pp. 212–227.

[4] Bagchi, S., Bai, X., Kalagnanam, J., 2006. Data quality management using business process modeling, in: IEEE SCC, IEEE Computer Society. pp. 398–405.

[5] Bai, X., Padman, R., Krishnan, R., 2007. A risk management approach to business process design, in: ICIS, Association for Information Systems. p. 28.

[6] Basel Committee on Bankin Supervision, 2006. Basel II - International Convergence of Capital Measurement and Capital Standards.

[7] Bergholtz, M., Grégoire, B., Johannesson, P., Schmitt, M., Wohed, P., Zdravkovic, J., 2005. Integrated methodology for linking business and process models with risk mitigation, in: REBNITA, Citeseer.

[8] Betz, S., Hickl, S., Oberweis, A., 2011. Risk-aware business process modeling and simulation using xml nets, in: Hofreiter, B., Dubois, E., Lin, K.J., Setzer, T., Godart, C., Proper,

E., Bodenstaff, L. (Eds.), CEC, IEEE. pp. 349–356.

[9] Bhuiyan, M., Islam, M.M.Z., Koliadis, G., Krishna, A., Ghose, A., 2007. Managing business process risk using rich organizational models, in: COMPSAC (2), IEEE Computer Society. pp. 509–520.

[10] Bhushan, N., Rai, K., 2004. Strategic Decision Making: Applying the Analytic Hierarchy Process. 3rd ed., Springer.

[11] Burton-Jones, A., Wand, Y., Weber, R., 2009. Guidelines for empirical evaluations of conceptual modeling grammars. Journal of the Association for Information Systems 10, 495–532.

[12] Commission, I.E., 1990. IEC 61025 Fault Tree Analysis (FTA).

[13] Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M.P., 2013. Supporting Risk-Informed Decisions during Business Process Execution. QUT ePrints 55979. Queensland University of Technology. http://eprints.qut.edu.au/55979.

[14] Conforti, R., Fortino, G., La Rosa, M., ter Hofstede, A.H.M., 2011. History-aware, real-time risk detection in business processes, in: Proc. of CoopIS, Springer.

[15] Conforti, R., ter Hofstede, A.H.M., La Rosa, M., Adams, M., 2012. Automated risk mitigation in business processes, in: Proc. of CoopIS, Springer.

[16] Cope, E.W., Küster, J.M., Etzweiler, D., 2009. Risk Extensions to the BPMN 1.1 Business Process Metamodel. Technical Report RZ3740. IBM Research.

[17] Cope, E.W., Küster, J.M., Etzweiler, D., Deleris, L.A., Ray, B., 2010. Incorporating risk into business process models. IBM Journal of Research and Development 54, 4.

[18] Cousins, P.D., Lamming, R.C., Bowen, F., 2004. The role of risk in environment-related supplier initiatives. International Journal of Operations & Production Management 24, 554–565.

[19] Davis, F.D., 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly 13, 319–340.

[20] Davis, R.B., Brabander, E., 2007. ARIS Design Platform: Getting Started with BPM. Springer.

[21] Dumas, M., van der Aalst, W.M., ter Hofstede, A.H., 2005. Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley & Sons.

[22] Fenz, S., 2010. From the resource to the business process risk level, in: Proceedings of the South African Information Security Multi-Conference (SAISMC'2010), pp. 100–109.

[23] Fenz, S., Neubauer, T., 2009. How to determine threat probabilities using ontologies and bayesian networks, in: Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies, ACM, New York, NY, USA. pp. 69:1–69:3.

[24] Gay, P., Pla, A., López, B., Meléndez, J., Meunier, R., 2010. Service workflow monitoring through complex event processing, in: ETFA, IEEE. pp. 1–4.

[25] Golden, W., Acton, T., Conboy, K., van der Heijden, H., Tuunainen, V.K. (Eds.), 2008. 16th European Conference on Information Systems, ECIS 2008, Galway, Ireland, 2008.

[26] Günther, C.W., Verbeek, E., 2013. Supporting Risk-Informed Decisions during Business Process Execution. Technical Report 55979. Eindhoven University of Technology. http://www.xes-standard.org/_media/openxes/openxesdeveloperguide-1.9.pdf.

[27] Harland, C., Brenchley, R., Walker, H., 2003. Risk in supply networks. Journal of Purchasing and Supply Management 9, 51 – 62.

[28] Hermosillo, G., Seinturier, L., Duchien, L., 2010. Using complex event processing for dynamic business process adaptation, in: IEEE SCC, IEEE Computer Society. pp. 466–473.

[29] Herrmann, P., Herrmann, G., 2006. Security requirement analysis of business processes. Electronic Commerce Research 6, 305–335.

[30] Hespos, R.F., Strassmann, P.A., 1965. Stochastic decision trees for the analysis of investment decisions. Management Science 11, 244–259.

[31] Hollingsworth, D., 1995. The Workflow Reference Model. Workflow Management Coalition. Workflow Management Coalition.

[32] Islam, M.M.Z., Bhuiyan, M., Krishna, A., Ghose, A., 2009. An integrated approach to managing business process risk using rich organizational models, in: Meersman, R., Dil-

lon, T.S., Herrero, P. (Eds.), OTM Conferences (1), Springer. pp. 273–285.

[33] Jakoubi, S., Goluch, G., Tjoa, S., Quirchmayr, G., 2008. Deriving resource requirements applying risk-aware business process modeling and simulation, in: [25]. pp. 1542–1554. pp. 1542–1554.

[34] Jallow, A.K., Majeed, B., Vergidis, K., Tiwari, A., Roy, R., 2007. Operational risk analysis in business processes. BT Technology Journal 25, 168–177.

[35] Johnson, W.G., 1973. MORT - The Management Oversight and Risk Tree. U.S. Atomic Energy Commission.

[36] Kaegi, M., Mock, R., Ziegler, R., Nibali, R., 2006. Information systems' risk analysis by agent-based modelling of business processes, in: Soares, C.G., Zio, E. (Eds.), ESREL, Taylor & Francis. pp. 2277–2284.

[37] Kang, B., Cho, N.W., Kang, S.H., 2009. Real-time risk measurement for business activity monitoring (bam). International Journal of Innovative Computing, Information and Control 5, 3647–3657.

[38] Karagiannis, D., Mylopoulos, J., Schwab, M., 2007. Business process-based regulation compliance: The case of the sarbanes-oxley act, in: RE, IEEE. pp. 315–321.

[39] La Rosa, M., ter Hofstede, A.H.M., Wohed, P., Reijers, H.A., Mendling, J., van der Aalst, W.M.P., 2011a. Managing process model complexity via concrete syntax modifications. Industrial Informatics, IEEE Transactions on 7, 255–265.

[40] La Rosa, M., Wohed, P., Mendling, J., ter Hofstede, A.H.M., Reijers, H.A., van der Aalst, W.M.P., 2011b. Managing process model complexity via abstract syntax modifications. Industrial Informatics, IEEE Transactions on 7, 614–629.

[41] Lambert, J.H., Jennings, R.K., Joshi, N.N., 2006. Integration of risk identification with business process models. Systems engineering 9, 187–198.

[42] Little, A., Best, P.J., 2003. A framework for separation of duties in an sap r/3 environment. Managerial Auditing Journal 18, 419–430.

[43] Lund, M.S., Solhaug, B., Stølen, K., 2011. Model-Driven Risk Analysis. Springer.

[44] Mendling, J., Strembeck, M., Recker, J., 2012. Factors of process model comprehension - findings from a series of experiments. Decision Support Systems 53, 195–206.

[45] Metropolis, N., 1987. The beginning of the monte carlo method. Los Alamos Science 15, 125–130.

[46] Meulbroek, L., 2000. Total strategies for company-wide risk control. Financial Times 9, 1–4.

[47] Meyer, B., 1990. Introduction to the theory of programming languages. Prentice-Hall.

[48] Mock, R., Corvo, M., 2005. Risk analysis of information systems by event process chains. International journal of critical infrastructures 1, 247–257.

[49] Neiger, D., Churilov, L., zur Muehlen, M., Rosemann, M., 2006. Integrating risks in business process models with value focused process engineering, in: Ljungberg, J., Andersson, M. (Eds.), ECIS, pp. 1606–1615.

[50] Object Management Group (OMG), 2011. Business Process Model and Notation (BPMN) ver. 2.0. Object Management Group (OMG). URL: `http://www.omg.org/spec/BPMN/2.0`.

[51] Oracle, http://download.oracle.com/docs/cd/E15523_01/integration.1111/e10224/bp_sensors.htm. Accesssed: June 2011. BPEL Process Manager Developer's Guide.

[52] Panayiotou, N.A., Oikonomitsios, S., Athanasiadou, C., Gayialis, S.P., . Risk assessment in virtual enterprise networks: A process-driven internal audit approach, in: Managing Risk in Virtual Enterprise Networks: Implementing Supply Chain Principles. IGI Global.

[53] Recker, J., 2010a. Continued use of process modeling grammars: the impact of individual difference factors. European Journal of Information Systems 19, 76–92.

[54] Recker, J., 2010b. Explaining usage of process modeling grammars: Comparing three theoretical models in the study of two grammars. Information & management 47, 316–324.

[55] Recker, J., 2010c. Opportunities and constraints: the current struggle with bpmn. Business Process Management Journal 16, 181–201.

[56] Recker, J., La Rosa, M., 2012. Understanding user differences in open-source workflow management system usage intentions. Information Systems 37, 200–212.

[57] Rosemann, M., zur Muehlen, M., 2005. Integrating risks in business process models, in: ACIS, AISeL.

[58] Rotaru, K., Wilkin, C., Churilov, L., Neiger, D., 2008. Formalising risk with value-focused process engineering, in: [25]. pp. 1583–1595. pp. 1583–1595.

[59] Rotaru, K., Wilkin, C., Churilov, L., Neiger, D., Ceglowski, A., 2011. Formalizing process-based risk with value-focused process engineering. Information Systems and e-Business Management 9, 447–474.

[60] Schmitt, M., Grégoire, B., Dubois, E., 2005. A risk based guide to business process design in inter-organizational business collaboration, in: International Workshop on Requirements Engineering for Business Need and IT Alignment (REBNITA 2005).

[61] Schwartz, P., 2000. When good companies do bad things. Strategy & Leadership 28, 4–11.

[62] Sienou, A., Karduck, A.P., Lamine, E., Pingaud, H., 2008. Business process and risk models enrichment: Considerations for business intelligence, in: ICEBE, pp. 732 –735.

[63] Sienou, A., Karduck, A.P., Pingaud, H., 2006. Towards a framework for integrating risk and business process management, in: Dolgui, A., Morel, G., Pereira, C.E. (Eds.), Information Control Problems in Manufacturing. Elsevier. volume 6, pp. 615–620.

[64] Sienou, A., Lamine, E., Pingaud, H., Karduck, A.P., 2009. Aspects of the bprim language for risk driven process engineering, in: Meersman, R., Herrero, P., Dillon, T.S. (Eds.), OTM Workshops, Springer. pp. 172–183.

[65] Sienou, A., Lamine, E., Pingaud, H., Karduck, A.P., 2010. Risk driven process engineering in digital ecosystems: Modelling risk, in: DEST, pp. 647–650.

[66] Simons, R.L., 1999. How risky is your company? Harvard Business Review 77, 85.

[67] Singh, P., Gelgi, F., Davulcu, H., Yau, S.S., Mukhopadhyay, S., 2008. A risk reduction framework for dynamic workflows, in: IEEE SCC (1), IEEE Computer Society. pp. 381–388.

[68] Smallman, C., 1996. Risk and organizational behaviour: a research model. Disaster Prevention and Management 5, 12–26.

[69] Standards Australia and Standards New Zealand, 2009. Standard AS/NZS ISO 31000.

[70] Strecker, S., Heise, D., Frank, U., 2010. RiskM: A multi-perspective modeling method for IT risk assessment. Information Systems Frontiers 13, 1–17.

[71] Suriadi, S., Weiß, B., Winkelmann, A., ter Hofstede, A., Wynn, M., Ouyang, C., Adams, M., Conforti, R., Fidge, C., La Rosa, M., Pika, A., 2012. Current Research in Risk-Aware Business Process Management - Overview, Comparison, and Gap Analysis. BPM Center Report BPM-12-13. BPMcenter.org.

[72] Sybase, http://www.sybase.com.au/files/White_Papers/Sybase_CEP_Implementation_Methodology_wp.pdf. Accessed: June 2011. Sybase CEP Implementation Methodology for Continuous Intelligence.

[73] Taylor, P., Godino, J.J., Majeed, B., 2008. Use of fuzzy reasoning in the simulation of risk events in business processes, in: Louca, L., Chrysanthou, Y., Oplatkova, Z., AlBegain, K. (Eds.), Proceedings of the 22nd European Conference on Modelling and Simulation, pp. 25–30.

[74] ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N., 2010. Modern Business Process Automation: YAWL and its Support Environment. Springer.

[75] Tjoa, S., Jakoubi, S., Quirchmayr, G., 2008. Enhancing business impact analysis and risk assessment applying a risk-aware business process modeling and simulation methodology, in: ARES, IEEE Computer Society. pp. 179–186.

[76] van Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P., 2008. Cycle time prediction: When will this case finally be finished?, in: Meersman, R., Tari, Z. (Eds.), OTM Conferences (1), Springer. pp. 319–336.

[77] Voluntary Interindustry Commerce Solutions Association, http://www.vics.org. Accessed: June 2011. Voluntary Inter-industry Commerce Standard (VICS).

[78] Wang, D., Rundensteiner, E.A., Ellison, R.T., Wang, H., 2011. Active complex event processing infrastructure: Monitoring and reacting to event streams, in: Abiteboul, S., Böhm, K., Koch, C., Tan, K.L. (Eds.), ICDE Workshops, IEEE. pp. 249–254.

[79] Weiß, B., Winkelmann, A., 2011. Developing a process-oriented notation for modeling operational risks - a conceptual metamodel approach to operational risk management in

knowledge intensive business processes within the financial industry, in: HICSS, IEEE Computer Society. pp. 1–10.

[80] zur Muehlen, M., Ho, D.T.Y., 2005. Risk management in the bpm lifecycle, in: Bussler, C., Haller, A. (Eds.), Business Process Management Workshops, pp. 454–466.

Appendix A. Actions

Here are listed all actions defined for the language.

Action	Description
(ID)	returns the ID of the generic instance that is being analyzed
[IDCurr]	returns the ID of the instance that the sensor is monitoring
Count	returns the number of times a task has been completed
offerResource	returns the resources to which the task has been offered
allocateResource	returns the resources to which the task has been allocated
startResource	returns the resources that started the task
completeResource	returns the resource that completed the task
isOfferd	returns "*true*" if the task has been offered
isAllocated	returns "*true*" if the task has been allocated
isStarted	returns "*true*" if the task has been started
isCompleted	returns "*true*" if the task has been completed
OfferTime	returns the time when the task has been offered
AllocateTime	returns the time when the task has been allocated
StartTime	returns the time when the task has been started
CompleteTime	returns the time when the task has been completed
OfferTimeInMillis	returns the time (in millisecond) when the task has been offered
AllocateTimeInMillis	returns the time (in millisecond) when the task has been allocated
StartTimeInMillis	returns the time (in millisecond) when the task has been started
CompleteTimeInMillis	returns the time (in millisecond) when the task has been completed
PassTimeInMillis	returns the amount of time (in millisecond) that was needed to complete the task
TimeEstimationInMillis	returns an estimation of the time (in millisecond) needed to completed the task/process
Variable	returns the value of the variable or sub-variable required
offerDistribution	returns the list of resources to which the task is offered by default
allocateDistribution	returns the list of resources to which the task is allocated by default
startDistribution	returns the list of resources to which the task is started by default
offerInitiator	returns the offering policy of the task (user or system)
allocateInitiator	returns the allocating policy of the task (user or system)
startInitiator	returns the starting policy of the task (user or system)
CountElements	returns the number of instances that satisfy the parameters required
FraudProbabilityFunc	returns the probability of a fraud using as parameters: the current number of executions, the maximum number of executions allowed, the parameter used to group these instances, the parameter used to identify a temporal window, and the dimension of the temporal window

List of actions of the architecture

Appendix B. Nested Loops

Here are listed the four type of loops defined for the language.

For	Description
forAND[][]	executes a nested loop for each list provided in input (among the first couple of brackets) resolving the expression (defined among the second couple of brackets), then returns the AND conjunction of the results obtained
forOR[][]	executes a nested loop for each list provided in input (among the first couple of brackets) resolving the expression (defined among the second couple of brackets), then returns the OR conjunction of the results obtained
forADD[][]	executes a nested loop for each list provided in input (among the first couple of brackets) resolving the expression (defined among the second couple of brackets), then returns the sum of the results obtained
forMUL[][]	executes a nested loop for each list provided in input (among the first couple of brackets) resolving the expression (defined among the second couple of brackets), then returns the product of the results obtained

List of nested loops

Appendix C. Functions

Here are listed all functions defined for the language.

Function	Description
offeredList	returns the list of tasks currently offered to the resource/resources (returns a list of list if used on resources)
allocatedList	returns the list of task currently allocated to the resource/resources (returns a list of list if used on resources)
startedList	returns the list of task currently started by the resource/resources (returns a list of list if used on resources)
offeredNumber	returns the number of tasks currently offered to the resource/resources (returns a list if used on resources)
allocatedNumber	returns the number of tasks currently allocated to the resource/resources (returns a list if used on resources)
startedNumber	returns the number of tasks currently started by the resource/resources (returns a list if used on resources)
offeredMinNumber	returns the minimum number of tasks currently offered to the resource/resources
allocatedMinNumber	returns the minimum number of tasks currently allocated to the resource/resources
startedMinNumber	returns the minimum number of tasks currently started by the resource/resources
offeredMinNumberExcept	returns the minimum number of tasks currently offered to the resource/resources excluding the resource/resources provided in input (the input is provided using a dotted format after the name of the function)
allocatedMinNumberExcept	returns the minimum number of tasks currently allocated to the resource/resources excluding the resource/resources provided in input (the input is provided using a dotted format after the name of the function)
startedMinNumberExcept	returns the minimum number of tasks currently started by the resource/resources excluding the resource/resources provided in input (the input is provided using a dotted format after the name of the function)
offeredContain	returns *true* if the task provided in input (using a dotted format after the name of the function) is currently offered to the resource/resources.
allocatedContain	returns *true* if the task provided in input (using a dotted format after the name of the function) is currently allocated to the resource/resources.
startedContain	returns *true* if the resource/resources started the task provided in input (using a dotted format after the name of the function).

List of functions

Appendix D. Questionnaire

Part 1) Background Questions

E1a: Which description matches best your current status?

- Student
- Academic
- Professional

E1b:Please specify your gender:

- Female
- Male
- Prefer not to tell.

E2: How many years ago did you start process modeling?

_____ Years

E3a: How many process models have you analyzed or read within the last 12 months? (A year has about 250 work days. In case you read one model per day, this would sum up to 250 models per year)

_____ Models

E3b: How many process models have you created or edited within the last 12 months?

_____ Models

E3c: How many activities did all these models have on average?

_____ Activities

E4a: How many work days of formal training on process modeling have you received within the last 12 months? (This includes e.g. university lectures, certification courses, training courses. 10 weeks of a 120 minute university lecture is roughly 3 work days)

_____Days

E4b: How many work days of self-education have you made within the last 12 months? (This includes e.g. learning-by-doing, self-study of textbooks or specifications)

_____Days

Part 2) Your experience with Workflow Management Systems

Q1: Which of the following (process) modelling techniques other than YAWL have you used to describe a process or procedure? Tick all that apply.

- None. Please go to question 10.
- BPMN
- UML
- Activity Diagrams
- EPCs
- BPEL
- Petri Nets
- Protos
- Other:

Q2: If you have used any technique other than YAWL, roughly, how many **conceptual process** models do you think you have created?

_____ process models

Q3: If you have used any technique other than YAWL, roughly, how many **workflow** models (i.e. executable process models) do you think you have read?

_____ workflow models

Q4: How long have you been using a Workflow Management System?

- I am evaluating to do so/I have just started
- Less than 1 month
- 1 - 6 months
- 7 - 12 months
- More than 1 year

Q6: Indicate your level of agreement to the following statements on the given scale by circling the number that best describes your view on the statement.

	Strongly disagree	Disagree	Somewhat disagree	Normal	Somewhat agree	Agree	Strongly agree
Overall, I am very familiar with Work flow Management Systems.	1	2	3	4	5	6	7
I feel very confident in my understanding of Workflow Management Systems.	1	2	3	4	5	6	7
I feel very competent in using Workflow Management Systems.	1	2	3	4	5	6	7

Part3) Your experience with the YAWL system

Q1: How long have you been using YAWL?

- I am evaluating to do so/I have just started
- Less than 1 month
- 1 - 6 months
- 7 - 12 months
- More than 1 year

Q2: Please indicate, roughly, the typical extent of usage of YAWL. This includes any activity related to the YAWL system, e.g. development, reading documentation, modelling, executing or simulating YAWL workflows, and using a system that is based on some YAWL component. Keep in mind, a regular working days has eight hours (480 minutes).

- Not applicable
- On average, I spend _____ hours and _____ minutes on YAWL every working day.

Q3: Roughly, how many YAWL models do you think you have created or read?

_____ YAWL models

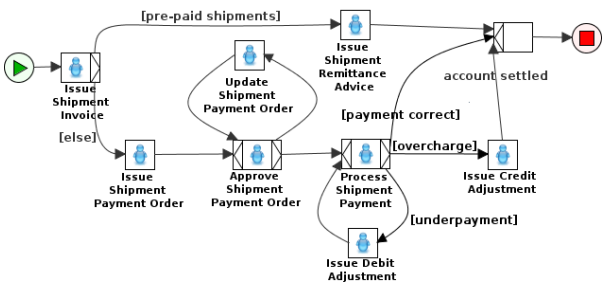
Q4: Which features of the YAWL system have you ever used? Tick all that apply

- Not applicable
- Execution environment

- Syntax checker/verification
- Cancelation region
- OR-join
- Multiple instance task
- Deferred choice
- Milestone
- Retain familiar/Separation of duties
- Deferred distribution
- Distribution set filter
- Allocation strategies
- Worklets/Exlets
- Custom services
- Configuration

Part 4) Risk Sensor Comprehension
In this part, you will be shown 3 risks related to a process described in YAWL. You will be asked questions about each of them.

Risk 1: Consider the following YAWL model and associated risk condition described below.



Variables:
A = case(current).Update Shipment Payment Order(Count)
B = case(Update Shipment Payment Order(Count)>=E).Update Shipment Payment Order(CountElements)
C = case(Update Shipment Payment Order(Count)>=A AND Process Shipment Payment(isOffered)="true"). Update Shipment Payment Order(CountElements)
D = 0.6
E = 5

Sensor Condition: ***(C/B)>D*** where"/" is the division operator

Q0. How difficult is it to understand the meaning of the above sensor condition:

- 1 – very simple
- 2 – rather simple
- 3 – neutral
- 4 – rather difficult
- 5 – very difficult

Q1. Does the sensor send a notification if the task Update Shipment Payment Order may not be executed?

- Yes / No / I do not know

Q2. Can the sensor notify a risk if the task Update Shipment Payment Order has been executed only twice?

- Yes / No / I do not know

Q3. Does A retrieve the value of a variable named Count of the task Update Shipment Payment Order?

- Yes / No / I do not know

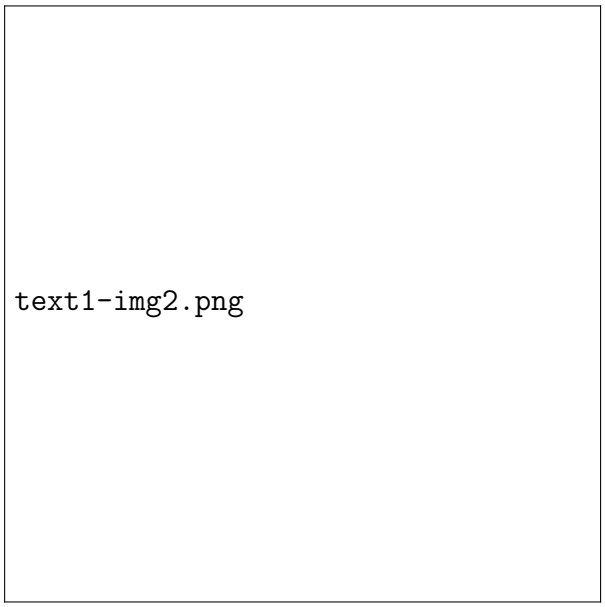
Q4.Does C return the number of instances where the task Process Shipment Payment has been offered?

- Yes / No / I do not know

Q5. Does D indicate a threshold that if exceeded produces a notification from the sensor?

- Yes / No / I do not know

Risk 2: Using the YAWL model of page 6 above, an Approval fraud risk has been identified using Fault Tree Analysis, as shown below.



To detect the risk of this fault, we first have to check whether there is an order, say order o of customer c, to be approved. This means checking that an instance of task Approve Shipment Payment Order is being executed. Moreover, we need to check that either of the following conditions holds:

1. o has been allocated to a Senior Finance Officer who has already approved another order for the same customer in the last d days; or
2. at least one Senior Finance Officer is available who approved an order for customer c in the last d days and all other Senior Finance Officers who never approved an order for c during the last d days are available

Q0. How difficult is it to understand the meaning of the above risk:

- 1 – very simple
- 2 – rather simple
- 3 – neutral
- 4 – rather difficult
- 5 – very difficult

Q1. Could condition Abe used as trigger for the sensor?

- Yes / No / I do not know

Q2. If condition B is expressed using a variable, would the following assignment be correct?
B = case(current).Approve Shipment Payment Order(allocateResource)

- Yes / No / I do not know

Q3. Could condition D be expressed using only one variable in the risk condition?

- Yes / No / I do not know

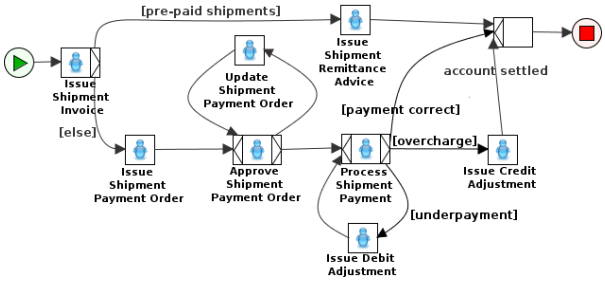
Q4. Could condition E be expressed using only one variable in the risk condition?

- Yes / No / I do not know

Q5. Could the AND between conditions B and C be expressed using only one variable in the risk condition?

- Yes / No / I do not know

Risk 3: Consider the following YAWL model and the associated risk condition described below.



Variables:

A	=	case(current).Process Shipment Payment.Customer
B	=	case(Process Shipment Payment. Customer=A AND Issue Debit Adjustment(isCompleted)="true"). Issue Debit Adjustment (CountElements)
C	=	case(Process Shipment Payment. Customer=A AND Issue Credit Adjustment(isCompleted)="true"). Issue Debit Adjustment (CountElements)
D	=	case(Process Shipment Payment. Customer=A).Process Shipment Payment(CountElements)
F	=	0.4

Sensor Condition:
(B/D)>F|(C/D)>F where"/" is the division operator and "|" is the logical OR operator

Q0. How difficult is it to understand the meaning of the above sensor condition:

- 1 – very simple
- 2 – rather simple
- 3 – neutral
- 4 – rather difficult
- 5 – very difficult

Q1. Does the sensor send a notification if the task Issue Debit Adjustment is not executed?

- Yes / No / I do not know

Q2. Will the sensor send a notification as soon as the task Issue Credit Adjustment is completed?

- Yes / No / I do not know

Q3. Does B return the number of instances where the customer is the same of the current instance and the task Issue Debit Adjustment has been completed?

- Yes / No / I do not know

Q4. Does C return the number of instances where the task Process Shipment Payment has been offered?

- Yes / No / I do not know

Q5. Does B always return a value greater than the value return by C?

- Yes / No / I do not know

Part 5) Your views on the use of the Sensor-based component of the YAWL system

This part of the survey captures some information about how you overall rate the Sensor-based component of the YAWL system you have been using. Please note again that all information you provide will be treated confidently. Thus, we ask you to please answer honestly.

In the following, you will be given a number of statements on opinions that you may have towards the Sensor-based

component of the YAWL system. Please indicate your level of agreement to the statements on the given scale by ticking the box that best describes your view on the respective statement.

Facilitating Conditions	Strongly disagree	Disagree	Somewhat disagree	Normal	Somewhat agree	Agree	Strongly agree
Guidance was available to me in the use of the Sensor-based component of the YAWL system.	1	2	3	4	5	6	7
Specialized instruction concerning the use of the Sensor-based component of the YAWL system was available to me.	1	2	3	4	5	6	7
A specific person or group was available for assistance with difficulties with the Sensor-based component of the YAWL system.	1	2	3	4	5	6	7
Satisfaction	Strongly disagree	Disagree	Somewhat disagree	Normal	Somewhat agree	Agree	Strongly agree
I am extremely pleased with my use of the Sensor-base component of the YAWL system.	1	2	3	4	5	6	7
I am extremely contented with my use of the Sensor-base component of the YAWL system.	1	2	3	4	5	6	7
I am extremely delighted with my use of the Sensor-base component of the YAWL system.	1	2	3	4	5	6	7
I am extremely satisfied with my use of the Sensor-base component of the YAWL system.	1	2	3	4	5	6	7

Perceived Usefulness	Strongly disagree	Disagree	Somewhat disagree	Normal	Somewhat agree	Agree	Strongly agree
Overall, I find the Sensor- based component of the YAWL system useful for modelling process-related risks.	1	2	3	4	5	6	7
I find the Sensor-based component of the YAWL system useful for achieving the purpose of modelling process-related risks.	1	2	3	4	5	6	7
I find the Sensor-based component of the YAWL system helps me in meeting my process-related risks modelling objectives.	1	2	3	4	5	6	7
Perceived Ease of Use	Strongly disagree	Disagree	Somewhat disagree	Normal	Somewhat agree	Agree	Strongly agree
I find it easy to model process-related risks in the way I intended using the Sensor-based component of the YAWL system.	1	2	3	4	5	6	7
I find learning to use the Sensor-based component of the YAWL system is easy.	1	2	3	4	5	6	7
I find easy to create process-related risks using the Sensor-based component of the YAWL system.	1	2	3	4	5	6	7

Intention to Use	Strongly disagree	Disagree	Somewhat disagree	Normal	Somewhat agree	Agree	Strongly agree
I intend to use the Sensor-based component of the YAWL system when I have to define and detect risks in business processes.	1	2	3	4	5	6	7
I predict I would use the Sensor-based component of the YAWL system.	1	2	3	4	5	6	7
I plan to use the Sensor-based component of the YAWL system in the future.	1	2	3	4	5	6	7
I prefer to continue to work with the Sensor-based component of the YAWL system.	1	2	3	4	5	6	7