Sensors and Guards in a Context-aware Workflow Environment

Nico Kerschbaumer and Johann Eder

Department of Informatics-Systems, University of Klagenfurt, A-9020 Klagenfurt, Austria

Abstract. We propose a system for integrating context awareness in workflow management systems. The context is captured via simple or complex sensors. The workflow management systems interacts with these sensors and can use data about the environment in making decisions about the control flow. Since the environment - and thus the context of workflow execution - may change, we introduce sensor guards which monitor the sensors during specified critical parts of the workflow execution and raise an exception when the data transmitted by sensors leave their expected ranges.

1 Introduction

Context-aware Workflows allow the use of context information to steer the control flow. There are several definitions of *context* in the literature. [1] defines context as "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*" and context-aware applications as "*A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*"

These context data is usually received through sensors and then processed in the workflow. A difference between context data and 'normal' data is that context data can only be read, but not updated by the WfMS. Both types of data are crucial for the correct execution of a workflow, because the control flow decisions depend on them. Normal data can be updated at any point within or outside of an active workflow instance. As a solution to this problem *Workflow Data Guards*[2] have been proposed which monitor the data in all guarded activities in a workflow and can detect violation of a predefined set of constraints. Data Guards do not prevent changes of data performed outside of the workflow, however they detect those changes and allow the WfMS to react accordingly.

Currently data guards are limited to monitor data that are retrieved from a database and used in a workflow. We extend that concept to generic sensors, which would allow the development of a context-aware workflow system. To illustrate how sensors and guards can be utilized imagine a simplified claim handling workflow of an Australian insurance company. The company handles claims according to a certain process, however when the storm season arrives claims are handled in a different way. If the company would use a context aware workflow system that utilizes sensors and guards,

Kerschbaumer N. and Eder J.
Sensors and Guards in a Context-aware Workflow Environment.
DOI: 10.5220/0003020900550064
In *Proceedings of the International Joint Workshop on Technologies for Context-Aware Business Process Management, Advanced Enterprise Architecture and Repositories and Recent Trends in SOA Based Information Systems (ICEIS 2010)*, page
ISBN: 978-989-8425-09-6

it would be possible for the company to be informed by the system when a good time to switch to a special storm season claim handling process arises. To detect the beginning of the storm season, the workflow system could use data measured by different sensor and calculate key values. The computation of key values to detect, e.g., lighting storms, is probably quite complex and needs to incorporate data measured over a period of time to deliver somewhat accurate results - thus we need sensors that have the capability to store their measurements. Basically the sensor guards will allow us to react to exceptions that occur in the environment. The environment is monitored with sensors and guards can detect exceptional situation based on measured sensor data. For example if the temperature and smoke concentration rises rapidly in a room, the system could detect a fire and and activate water sprinklers.

We represent workflows as full-blocked workflow graphs and explore the concept of sensor guards in detail for this workflow model. Full-blocked workflow graphs are directed acyclic graphs(DAG) with two kinds of nodes: activities and control nodes. The edges determine the execution sequence of nodes. Control nodes describe basic workflow control structures: conditional(conditional-split and conditional-join nodes symbolized by a circle with *CS* or *CJ*) and parallel execution (par-split and par-join nodes symbolized by a circle with *PS* or *PJ*). In a full-blocked workflow graph, with each split node is associated exactly one join node of the same type and each join node has exactly one corresponding split node of the same type. The choice of full blocked workflows might be seen as a restriction, however full-blocked workflows are less problematic than non-blocked workflows, as it is does not allow process definitions that suffer from problems like deadlocks during execution and [3] shows that non-blocked workflows can get problematic as soon as data-flow comes into play. The approach we present in this paper is an extension of data guards[2] to include generic sensors that will help to overcome the stated problems.
The remainder of this paper is structured as follows: Section 2 gives an overview of related work about context-aware workflow systems and the data guard concept. In section 3 we introduce the concept of sensors, sensor guards, sensor broker and our meta model followed by section 4, which summarizes the paper and draws some conclusion.

2 Related Work

An early work in the area of context-awareness was the Context Toolkit [4] which introduced software components to query sensor data. In contrast to workflows, context data was used in pervasive and interactive computing for quite some time. In the recent years the research in context-aware workflows got a lot of momentum. [5] uses context information for the management of Webservice composed applications. In [6, 7] the authors present how context is managed in the Nexus project and introduce their vision of a smart factory which uses sensor technology to create a context-aware production process. [8] propose a methodology for context-sensitive business process development. Their approach supports modeling of context sensitive workflow regions which behave differently depending on the context information. Furthermore they introduced context change patterns that allow one to identify context changes that may influence the behavior of the process. The authors of [9] introduce a language called *Ubiquitious Workflow*

Definition Language(uWDL) that supports the specification of context information in workflow and evaluated it's feasibility in a workflow system called *uFlow*. [10] introduced complex event processing(CEP) which they define as a set of techniques and tools to understand and control event driven information systems. Complex events are events that only happen if a set of other events occur and can be related in various ways, by cause or by timing. CEP can be utilized in a broad spectrum of information systems like e.g. for business process automation, scheduling and control, network monitoring and performance prediction etc. The authors of [2] introduced workflow data guards which are constraints defined over specified parts of a workflow. The guard is specified in a workflow definition, consists of a condition, a set of activities that activate the guard for the first time in a workflow instance and a set of guarded activities. The condition is a boolean expression which is defined on the data used in the workflow and checked during the execution of a guarded activity. The sensor guards are an extension to workflow data guards by [2] and use methods and basics how to build context aware workflow systems as in [6, 7, 5, 8]. Furthermore for the realization of complex sensors techniques developed in the field of complex event processing [10] will be utilized. We also want to support switching to different process variants based on sensor and thus want to refer to a quite recent work by [11] who introduced criteria for guaranteeing soundness in configurable process variants. Lastly another important part is the support of exception handling and migration mechanisms with our sensor guards [12, 13].

3 Workflow Sensor Guards

A workflow sensor guard is a constraint defined over a specified part of a workflow like the ordinary data guard[2], however the constraints are not limited to the monitoring of some local variables, but can include data from sensors, services or events. Before we present the details of the sensor guard concept in Section 3.3 we introduce at first data guards in Section 3.1, our sensor concept and the different types in Section 3.2. In Section 3.4 and Section 3.5 we propose our sensor- and sensor guard broker, followed by a discussion of our metamodel in Section 3.6 and lastly we close with a brief overview how sensor guards can be utilized for exception handling in Section 3.7.

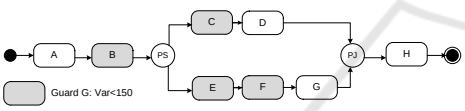
3.1 Preliminaries

The grounding of the sensor guard is the data guard [2] thus we give at first a quick overview of the original guard concept and then follow up with the extension that come with sensor guards. A data guard instance is in one of three states during runtime: inactive, active and deactivated. A guard is inactive when the workflow instance is started, the first guarded activity sets a guard to the state active. During the execution of an instance the guard can be deactivated and reactivated again, so that it is only active during execution of guarded activities. For details of the activation policy of a guard refer to [2].

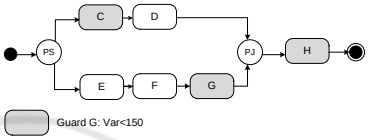
The activation mechanism allows designers to choose activities and paths where guard constraints are checked, thus minimizing performance problems. The deactivation of data guards is not straight forward, because parallel splits and conditional splits add the following problems that have to be solved at runtime:

- Tracking several guarded paths that run in parallel and provide communication between them.
- Detecting that a given guarded path has completed.
- It is not always possible to decide using only the workflow definition which successor will actually be started, i.e. after a conditional split node there exist different execution paths that can be taken. That leads to the problem that a guarded activity instance may have a guarded direct activity successor in one instance graph and none in another instance graph. For example refer to figure 1(c), considering only the workflow definition it is not possible to tell if the successor of B will be C or D, and thus it is not possible to determine if the guard can be deactivated after the completion of B.

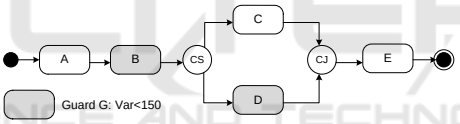
A guarded path may begin before or after a parallel split and continue after the parallel join as shown in figure 1(a) and figure 1(b) respectively. Figure 1(a) shows two parallel guarded paths, the guard G1 is started when an instance of activity B is started and stays active until both guarded paths, i.e. the path (B,PS,C) and the path (B,E,F).



(a) Workflow graph with parallel guarded paths



(b) Workflow graph with parallel guarded paths



(c) Workflow graph, in which it is unknown if an instance of B will have a guarded direct activity successor at runtime

Fig. 1. Example workflows with data guards cf.[2].

3.2 Sensors and Sensor Types

In general, a sensor is any device that measures a (physical) quantity and converts it into a signal which can be read by an observer, an instrument or even a computer that could offer the signal via a Webservice. Nowadays sensors are used in a wide variety of fields, from simple sensors that measure temperature or humidity to touch sensitive buttons on lamps that control the light intensity. Software sensors can monitor any data e.g. from websites, news feeds, or databases. In our model a sensor is composed of a unique id, an update interval, a query method and two methods to activate or deactivate the sensor. The value returned by the query method is the current measurement of a sensor. We symbolize sensor nodes as ”S” surrounded by two circles. Sensors can be queried from an activity, a decision node which can use sensor data in its condition or any other application that is capable of invoking a Webservice. Figure 3 shows an

example workflow were the outcome of the conditional split nodes CS depends on the value returned by the sensor *Simple*. Any activities like in our example B are also able to receive sensor data as a input. Based on this general sensor we derive four subtypes:

Simple Sensor. A simple sensor has a query function which allows one to read the data that the sensor currently measures. The update interval says how often the sensor updates its data.

Complex Sensor. A complex sensor is capable of hooking simple sensors, that means it can read and store the data of multiple simple sensors. Hooked sensors are stored in an array and the sensor data can be stored in a log. An entry in the data log contains a date-time stamp, the measured data and the sensors id. The stored measurements allow calculation of important coefficient or data aggregates which produce results that are a lot more accurate than a single sensor measurement. For a recent work on concerning data quality in context-aware workflow systems we would like to refer the reader to [14].

Active Sensor. An active sensor is a subtype of a complex sensor which features an event subscription mechanism. The conditions and constraints of a passive sensor guard are checked whenever a guarded node instance is activated. In contrast, an active sensor offers a set of events that a sensor guard can subscribe to and if the constraint fails to hold at any time, all sensor guard instances that subscribed to the active sensors event will be notified. The offered events use sensor data received from hooked simple sensors as a basis to perform calculations and checks that are then exposed as subscribable events.

Passive Sensor. This sensor type does not offer a single query function like the simple sensors, however it maintains a set of Webservices that can be called by sensor guards. The stored data in the log allows the passive sensor to offer services that for example calculate aggregates over past data or check if a condition was present during a certain date-time interval. A published service has a URL, a name and a query function that returns a value calculated by the service.

3.3 Sensor Guards

A sensor guard is specified in the workflow definition and consists of a boolean condition defined with data used in the workflow, measurements received from sensors, results from service calls that are offered by passive sensors or the occurrence of an event offered by an active sensor. The use of sensors allows the creation of context-aware workflows which can detect constraint violations via the predefined sensor guards. We extend our workflow model with the functionality to query sensors and utilize the measured data in activity steps or conditionals, similar to [6]. When sensors are queried, their result is stored in a local workflow variable which can then be evaluated at a conditional node or further processed in a workflow step that receives the data as an input. It is also possible to query sensors from a conditional node and evaluate the values directly. However, this approach is limited to simple constraints that will only evaluate a single data measurement of a sensor and is not capable of handling complex constraints that include for example aggregates of stored sensor data or precalculated values. To overcome the limits we introduced different types of sensors that provide functions exposed as Webservices which can be used in a guards constraint and events to which

a guard can subscribe. Sensors and sensor guards are managed by brokers which are described in detail in Section 3.4 and Section 3.5. Due to space limitations we are not able to present details how to incorporate and implement the sensor and guard concepts, however we want to sketch shortly how an sensor can be used in conjunction with WS-BPEL. Since simple and complex sensors are web services, they can be easily integrated and used in a for example a case condition which is evaluated from the BPEL engine and the result is depending on the return value of the sensor. Based on the outcome of the condition an exception handling routine can be started with fault handlers and throw activities. An active sensor could be implemented with a parallel branch in the BPEL process that listens for messages of the sensor guard broker and sets for example a boolean variable which will indicate if an event monitored by the external active sensor occurred.

3.4 Sensor Guard Broker

A workflow designer can create sensor guards at the sensor guard broker. The broker manages all guards, that means it activates and deactivates them depending on their activation set. The sensor guard broker provides the following methods:

- CreateGuard(gid, constraint, activation set, subscription set): Creates a new sensor guard with a constraint an optional activation and/or subscription sets. The activation set denotes the set of activities in which the guard instance will be activated. The subscription set, contains a list of subscribed active sensor events, which can be used in the guards constraint.
- RemoveGuard(gid): Removes a sensor guard from the broker. This is only possible if there exist no running instances of a guard.

A sensor guard instance has a unique gid, a constraint, a state (activated, deactivated), an activation set, a subscription set and provides the following functions:

- Activate() / Deactivate(): Called by the broker to enable or disable a sensor guard instance during the runtime of a workflow.
- Subscribe(URI, SubConstraint): Creates a subscription for an event provided by an active sensor. The subscribed event can then be used in a guards constraint as part of the boolean condition since the event either occurred, i.e. its true or it did not occur, i.e. its false.
- Unsubscribe(URI): Removes a subscription of a sensor guard at the sensor broker.

3.5 Sensor Broker

A sensor broker can be seen as a central repository that manages and controls sensors. A sensor has to be registered at the sensor broker, after that the sensor can be found and used via the broker. A workflow designer can register sensors at the service broker and sensor guards can then query simple sensors directly, use services provided by a passive sensor or subscribe to events offered by an active sensor. The registered sensors are reusable and each guard instance can include an arbitrary number of constraints defined over a set of available sensors, services and events. The broker provides the following methods:

- RegisterSensor(SensorId): Registers a sensor at the broker. Once a sensor has been registered it can be used by a sensor guard.
- UnregisterSensor(SensorId): Removes a sensor from the broker.

3.6 Meta Model

We use a workflow metamodel shown in figure 2 to describe workflow definitions, workflow instances, sensors, the sensor broker and sensor guards. A workflow consists of nodes and each node has a unique identifier and a type (activity, conditional split, parallel split). A node instance can read data from simple sensors or call services offered by passive sensors and use them in conditional split nodes for a decision or in an activity. Furthermore a node can have adjacent successors and predecessor. A workflow may also contain sensor guards which are an extension of data guards. In contrast to a data guard, a sensor guard is not limited to monitor workflow data inside a database, it can also monitor external sensors. We support three kinds of sensors namely simple-, complex, active- and passive sensors. A sensor guard instance can query simple sensors, call services offered by a passive sensor or subscribe to events provided by an active sensor. A service of a passive sensor is a Webservice that can be perform any type of computation with sensor data. An event provided by an active sensor monitors sensor data and notifies all subscribers when an event occurs.

Figure 3 shows an example workflow with a sensor guard that uses all introduced types of sensors in its guard constraint. Activities that are shaded in grey are guarded activities and only during execution of this activities the sensor guard will be active and inform the user or engine about a violation of its constraint. The workflow contains a simple sensor(Simple), a passive sensor(Passive) and an active sensor(Active) and an example constraint that uses the given sensor like, e.g., $Simple < 30 \;\&\&\; Passive.Some{-}Service() > 20 \;\&\&\; !Active.StormWarningEvent()$. Simple sensors can be queried by any activity node or conditional node and be used in guard constraints. Services offered by the passive sensor can be used in activities, conditionals and guards, for example activity A calls the GetHumidity() Webservice and the guard constraints also contains a service call to the passive sensor in its constraints. Events offered by active sensors can be subscribed by sensor guards and then used in their constraint. If an event occurs between transitions of two activities, then the next time the guard becomes active he receives a notification of the corresponding active sensor and the guard constraint will invalidate. If any part of the boolean constraint is violated while the guard is active then either a workflow administrator is informed or an exception handling routing is started.

3.7 Exception Handling

Based on sensor guards, that monitor that certain predefined conditions hold during a workflow instance execution common exception handling and compensation methods can be applied. For example *rollback* to a certain point in the workflow, *migrate* all running instances to a new workflow schema, *migrate-adhoc* only a single running instance to a new ad-hoc workflow, *suspend-resume* which allows to suspend a workflow instance and resume it at a later point when for example an exception that was raised by

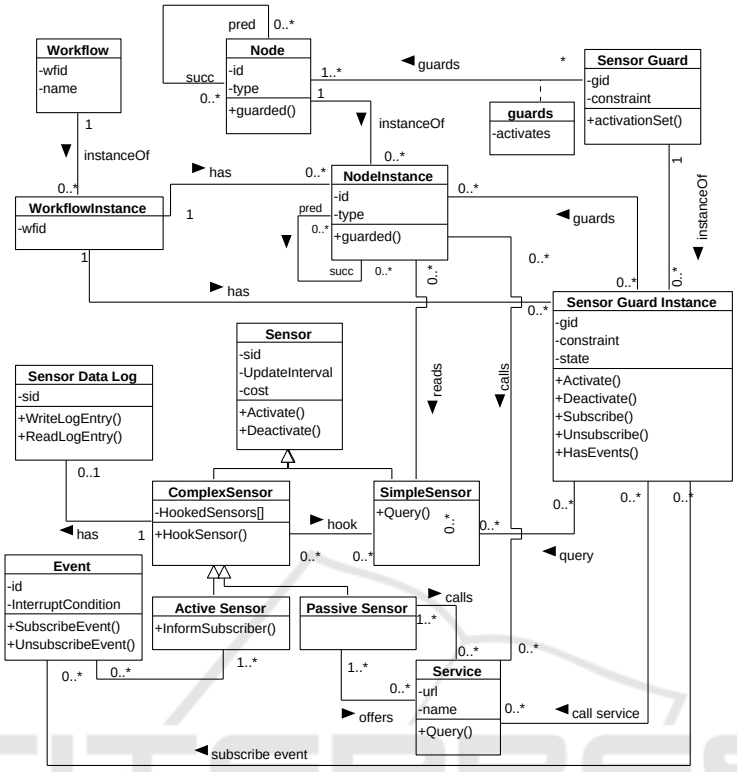


Fig. 2. General Workflow Metamodel incorporating Sensor Guards.

a sensor guard during the execution of an activity step will be resolved with a high probability and lastly *rollback and retry*. For some details on workflow exception handling refer to [12], for workflow evolution we'd like to refer the reader to [13] which features a very mature framework and describes methods that support instance migrations and ad-hoc instance changes. The example of the suspend and resume method shows, that there is also a need for intelligent schedules for sensors, that can calculate in advance when a sensor has to be switched on to collect data, so that a workflow instance that uses a service that needs logged data for a certain time period to work.

4 Conclusions

We introduced an architecture that supports the modeling of sensor-aware workflows and discussed techniques that are needed to implement our approach. Sensors offer workflow management systems information about the environment and allow the execution of workflow to depend on the context of the process execution. Since the context may change during the execution we introduced workflow sensor guards which are an extension to the already powerful mechanism of data guards for the treatment of data aspects in WfMSs. Guards can be defined in the workflow definition and are enabled to

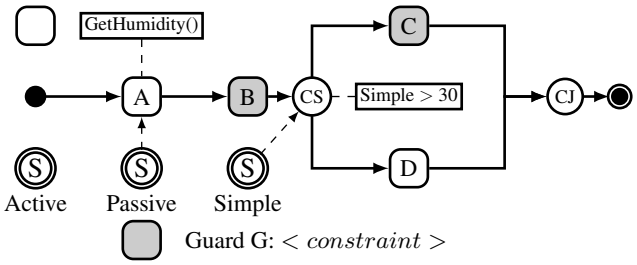


Fig. 3. Workflow with Sensor Guard.

protect the workflow from unrecognized changes in the context. When a sensor value leaves the range of expected values the guard raises an exception and the workflow system can react with the usual exception handling mechanisms. So sensors and guards allow the creation of context-aware workflows and support the monitoring and steering of the control flow through data provided by sensors.

References

1. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. 5 (2001) 4–7
2. Eder, J., Lehmann, M.: Workflow data guards. In: OTM Conferences (1). (2005)
3. Combi, C., Gambini, M.: Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data. In: Proc. CoopIS 2009. Volume 5870 of LNCS., Springer (2009)
4. Salber, D., Dey, A.K., Abowd, G.D.: The context toolkit: aiding the development of context-enabled applications. In: CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems. (1999)
5. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: A framework for the management of context-aware workflow systems. Proc. of WEBIST 2007, 3rd Int. Conf. on Web Information Systems and Technologies (2007)
6. Wieland, M., Kopp, O., Nicklas, D., Leymann, F.: Towards Context-Aware Workflows. In: CAiSE07 Proceedings of the Workshops and Doctoral Consortium Vol.2, 2007, Tapir Academic Press (2007)
7. Wieland, M., Kaczmarczyk, P., Nicklas, D.: Context Integration for Smart Workflows. In: Proceedings of the Sixth Annual IEEE Int. Conf. on Pervasive Computing and Communications, IEEE computer society (2008)
8. Modafferi, S., Benatallah, B., Casati, F., Pernici, B.: A methodology for designing and managing context-aware workflows. In: Mobile Information Systems II. (2005)
9. Shin, K., Cho, Y., Choi, J., Yoo, C.W.: A workflow language for context-aware services. In: MUE '07. Multimedia and Ubiquitous Engineering, 2007. (2007)
10. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
11. Hallerbach, A., Bauer, T., Reichert, M.: Guaranteeing soundness of configurable process variants in provop. In: CEC '09: Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing. (2009)
12. Eder, J., Liebhart, W.: The workflow activity model wamo. In: Third International Conference on Cooperative Information Systems (CoopIS 1995). (1995)

13. Dadam, P., Reichert, M.: The adept project: a decade of research and development for robust and flexible process support. Computer Science - R&D 23 (2009) 81–97
14. Wieland, M., Käppeler, U.P., Levi, P., Leymann, F., Nicklas, D.: Towards Integration of Uncertain Sensor Data into Context-aware Workflows. In: Tagungsband INFORMATIK 2009. (2009)