Process Monitoring Using Sensors in YAWL

Raffaele Conforti[1], Marcello La Rosa[1,2], and Giancarlo Fortino[3]

[1] Queensland University of Technology, Australia
{raffaele.conforti,m.larosa}@qut.edu.au
[2] NICTA Queensland Lab, Australia
[3] Università della Calabria, Italy
g.fortino@unical.it

Abstract. This article describes the architecture of a monitoring component for the YAWL system. The architecture proposed is based on sensors and it is realized as a YAWL service to have perfect integration with the YAWL systems. The architecture proposed is generic and applicable in different contextes of business process monitoring. Finally, it was tested and evaluated in the context of risk monitoring for business processes.

1 Introduction

The growing number of cases in which workflow management systems are utilized to execute business processes, created the need for companies to monitor the execution of their process instances [4]. Being able to monitor a process instance is a basic requirements for companies that want to prevent the eventuation of risks, be aware of the processing cost of process instances, or simply verify if a process instance is being delayed.

Several commercial workflow management systems already provide monitoring functionality for their systems, e.g. WebSphere[4], Oracle BAM[5], Sybase[6]. This type of functionality is not yet available in open-source workflow management systems, such as the YAWL workflow management system.[7]

In this article we will illustrate how to realize a multi-purpose monitoring component for the YAWL system. The component will be realized as a service for the YAWL system, to guarantee a perfect integration with the system.

This article is structured as follows. Section 2 provides a briefly description of the YAWL systems. Section 3 shows how to realize the monitoring component, which is evaluated in Section 4. Section 5 discusses related work and finally Section 6 concludes the article.

[4] http://www-142.ibm.com/software/products/au/en/subcategory/SW920
[5] http://www.oracle.com/technetwork/middleware/bam/overview/index.html
[6] http://www.sybase.com.au/products/financialservicessolutions/complex-event-processing
[7] http://www.yawlfoundation.org/

2 Requirements and Preliminaries

In order to monitor a business process, being able to have a complete overview of process instances is a requirements. The status of a process instance is fully provided by information about instances of tasks (work items) and subprocesses (nets) belonging to such a process instance.

When we consider a work item, to properly describe its status is essential to know: i) if the work item was performed or not (status in the life-cycle of a work item); ii) who (resource) performed the work item; iii) when the work item was performed (time stamp); and iv) how it was performed (data). A similar set of information is required for an instance of a net, except for the resource.

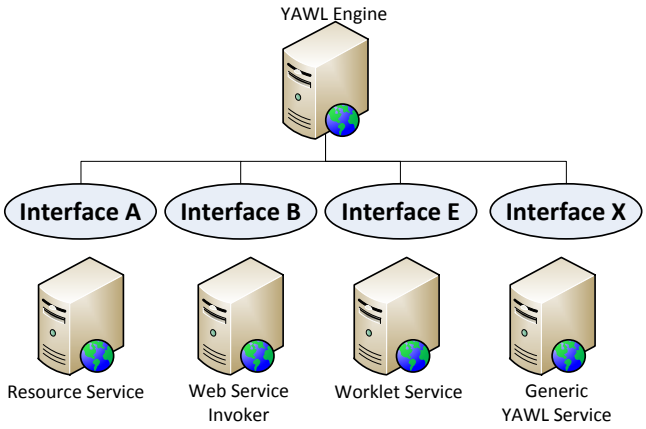

Fig. 1. YAWL Architecture [8]

A clear understanding of the YAWL environment [8] is required if we want to be able to access the status of work items and nets. The YAWL system is a service-oriented architecture built using Java. Figure 1 provides a simplified overview of the architecture of the YAWL system. The core element of this architecture is the YAWL engine, it is in charge of managing the instantiation of process instances and work items. The engine provides four interfaces to allow services to interact with the engine, for example allowing the resource service, which manages resources, to perform a work item instantiated.

These four interfaces are: i) interface A, which provides connection capabilities and allows process models to be uploaded and unloaded, and external services to be registered and unregistered; ii) interface B, which allows process instances to be lunched, work items to be checked-out for their execution, and process information to be retrieved; iii) interface E, which allows process logs to be retrieved; and iv) interface X, which provides a terminal for detecting and handling exceptions.

3 Monitoring Component

We can now describe how to create a monitoring component for the YAWL system, as the one realized for [1]. The best way to realize a new component for the YAWL system is to realize it as a YAWL service. Our service will use the interfaces provided by the system, described in section 2, to receive notifications from the engine about the initialization of the system and the starting of new instances.

Two are the interfaces that will provide information that may be relevant for us, they are interface B and interface X. The first interface will notify us with events related with the life-cycle of a process instance and the initialization of the YAWL system. Interface X will notify us when a process instance is canceled, and a case is enabled or completed.

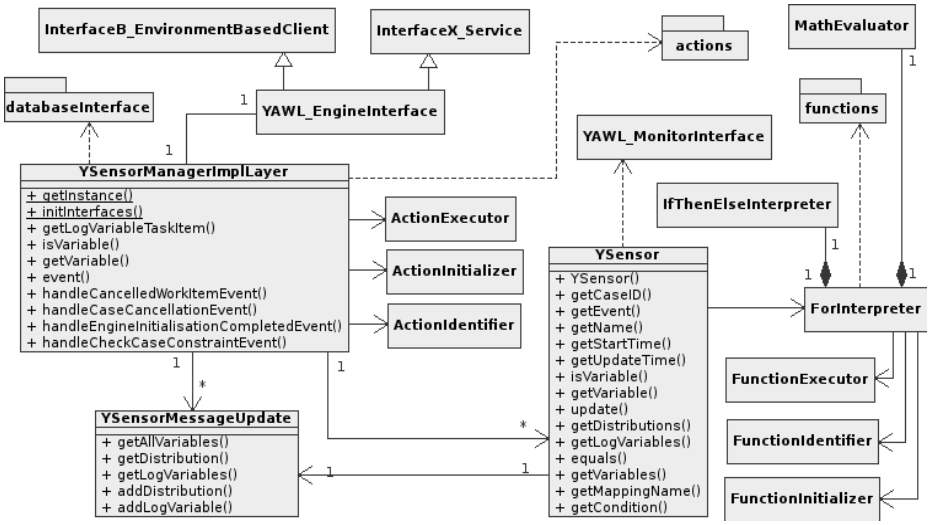

Fig. 2. Monitor Component: Class Diagram

In order to receive the notifications sent by these two interfaces, our monitoring component must extend the java class *InterfaceBWebsideController* and implements the java interface *InterfaceX_Service*. We are only interested in a subset of all notifications that interface B and X will send, for this reason we only need to implements these four methods: i) *handleCancelledWorkItemEvent*; ii) *handleCaseCancellationEvent*; iii) *handleEngineInitialisationCompletedEvent*; and iv) *handleCheckCaseConstraintEvent*. Figure 2 shows the UML [5] class diagram of how our monitoring component is built. In the diagram are only shown the classes required for the realization of the monitoring component and the methods that are relevant for us.

Our monitoring component works through the use of sensors (*YSensor*). Each sensor monitors a specific condition, composed of variables that are the result of functions and actions, and is managed by the sensor manager (*YSensorManagerImplLayer*). The sensor manager manages the creation of sensors and notifies them when changes occur in a process instance. The sensor manager creates new sensors each time a new process instances is started. Information about the initialization of the system and the starting, completion and cancellation of process instances are retrieved by the sensor manager using the *YAWL_EngineInterface*.

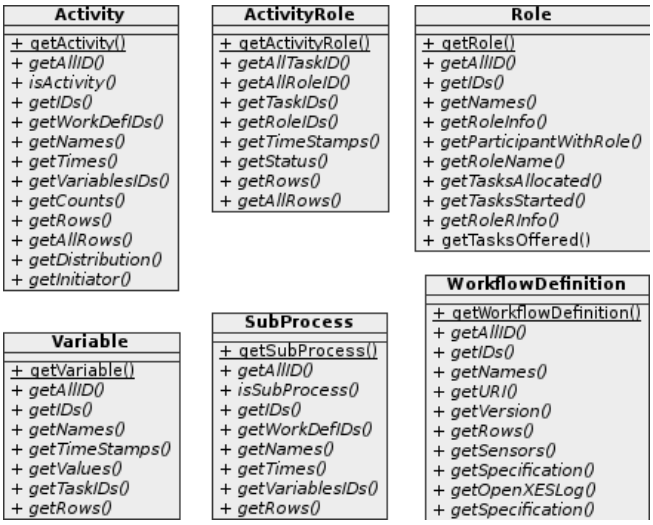

Fig. 3. DatabaseInteface Package Class Diagram

Changes in the process instance are discovered using the classes provided by the package *databaseInterface* (see Figure 3). This package provides an abstraction layer on top of the database which allows queries to be executed through the invocation of java methods. Executing queries gives to the system the possibility of retrieving information from different cases, and then have monitoring conditions defined across cases. In order to know which changes are relevant for a sensor, the sensors manager retrieves from it the list of *LogVariables*. Each of this logVariable is associated with an action that using the *ActionIdentifier* and the *ActionExecutor* is identified and executed, retrieving from the log the information of interest. Each action captures a specific aspect of a work item or a net, the list of all possible actions is shown in figure 4.

Changes in a process instance are then notified to a sensor using messages (*YSensorMessageUpdate*). Every time a sensor receives a message, it checks its monitoring condition using the updated information. The condition is checked using a specific interpreter (*ForInterpreter*) which interprets the languages defined in [1] and verifies if the condition is violated or not. In case the condition

is violated the sensor will notify the administrator by sending a notification through the *YAWL_MonitorInterface*. The condition that a sensor can monitor is a boolean expression which may contain nested loops, if-then-else constructs, and algebraic operators.

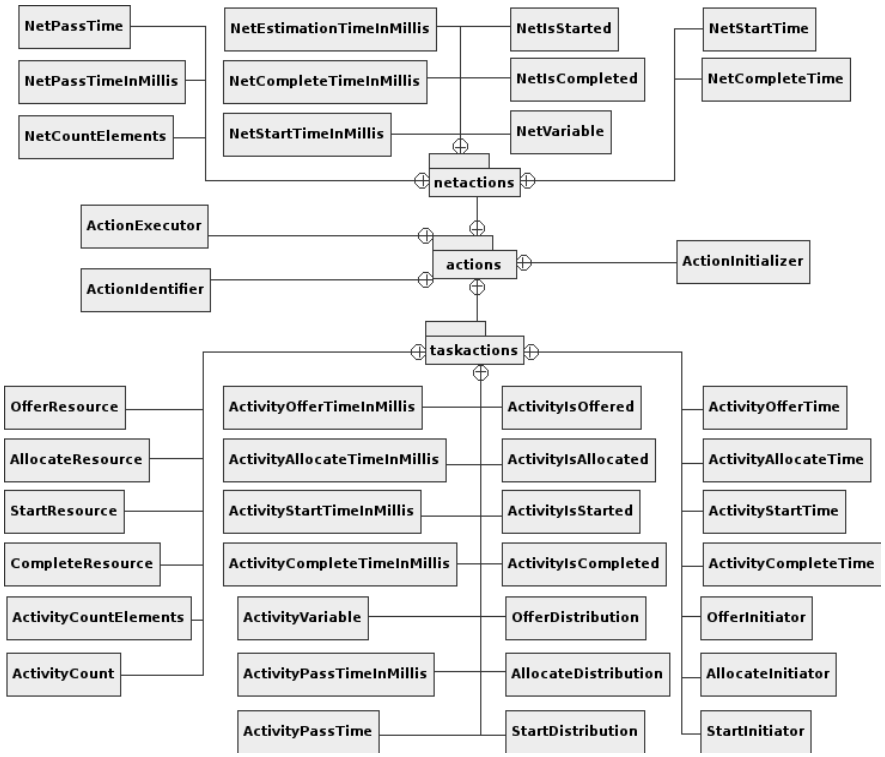

Fig. 4. Actions Package Class Diagram

4 Evaluation and Possible Uses

The architecture here proposed was evaluated in [1]. In the experiment we measured the time required to retrieve the result of the execution of an action. In table 1 we show the result grouped for type of action, e.g. NetStartTime and NetCompleteTime are grouped under the name NetXTime.

The results show that in general a result is produced in few milliseconds (ca. 20ms). Retrieving a net or an activity variable and retrieving information about the distribution set and the initiator of an activity require more time, this because they require the parsing of XML strings since the data are not directly stored in the database.

Actions	**Description**	**time [ms]**
NetIsX	functions checking if a net status has been reached	18.9
NetXTime NetXTimeInMillis	functions returning the time when a net status has been reached	18.8
NetVariable	returns the value of a net variable	432.6
ActivityCount	number of times a task has been completed	19.8
XResource	functions that return the resources associated with a task	20.9
ActivityIsX	functions checking if a task status has been reached	30.5
ActivityXTime ActivityXTimeInMillis	functions returning the time when a task status has been reached	22.3
ActivityVariable	returns the value of a task variable	96.7
XDistribution	functions returning the resources associated with a task by default	243
XInitiator	functions returning the allocation strategy for a resource association	249.6

Table 1. Performance of basic functions.

Risk monitoring is not the only context in which our component can be used. It is a multi-purpose monitoring component that provides the possibility of adding new actions in order to make it usable in other context such as for example cost monitoring, resource monitoring, or time monitoring.

5 Related Work

The idea of monitoring business processes is not new in the area of business process management. Academics explored the possibility of monitoring business processes using Complex Event Processing (CEP) systems [2, 3]. Commercial workflow management systems in general provide integrated monitoring features, e.g. Oracle Business Activity Monitoring (BAM) [6], webMethods Business Events[8], and SAP Sybase [7].

The monitoring component here discussed was used in the approach proposed in [1] for risk monitoring. In this approach business processes are integrated with aspects of risk management, specifically risk monitoring. Risk conditions are composed of two elements, a risk likelihood which monitors the likelihood of a risk to occur and a risk threshold which defined level of risk which the company is willing to accept before detecting the eventuation of a risk.

Gay et al. [2] propose the use of complex event processing for workflow monitoring on Petri nets. They identify six events that can represent the basic activities that a workflow can perform (i.e. Transition activation, Resource allocation, Resource liberation, Advance token, Start workflow, and End workflow). Using these simple events they have created six complex events that represent unwanted situations: i) Lack of resource; ii) Activity delay; iii) Lack of resource delay; iv) Transition delay; v) Workflow delay; vi) Interruption warning. This

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

approach, compare to our approach, does not take in consideration the data prospective. This limitation produces as consequence the possibility of defining conditions that are mainly related to the performance of a process instance.

Finally, Hermosillo et al. [3] propose a framework for dynamic business process adaptation in the context of BPEL processes. In this approach they use the monitoring functionality obtained using a CEP engine to detect conditions that will then trigger an adaptation, i.e. the add or change of a service.

6 Conclusion

In this article we showed how to realize a monitoring component for the YAWL system using the interfaces provided by the system itself. The monitoring is done using sensors, which monitor conditions that can be defined using information across cases.

The main contribution of this work is the identification and documentation of a minimal set of classes required for the realization of a monitoring component for the YAWL system.

The component is realized as a custom YAWL service, in order to guarantee a perfect integration with the YAWL system. The structure of the component also results to be independent from a specific monitoring purpose, consenting its application in different contexts could they be risk monitoring, or cost monitoring.

Finally, the architecture was implemented and tested. The results of the test show that the retrieving of information can be computed efficiently and without requiring additional work engine side.

References

1. R. Conforti, G. Fortino, M. La Rosa, and A. H. M. ter Hofstede. History-aware, real-time risk detection in business processes. In *Proc. of CoopIS*, volume 7044 of *LNCS*. Springer, 2011.
2. P. Gay, A. Pla, B. López, J. Meléndez, and R. Meunier. Service workflow monitoring through complex event processing. In *Proc. of ETFA*, pages 1–4. IEEE, 2010.
3. G. Hermosillo, L. Seinturier, and L. Duchien. Using complex event processing for dynamic business process adaptation. In *Proc. of IEEE SCC*, pages 466–473. IEEE Computer Society, 2010.
4. A. Kronz. Managing of process key performance indicators as part of the aris methodology. In *Corporate Performance Management*, pages 31–44. Springer, 2006.
5. Object Management Group (OMG). *OMG Unified Modeling LanguageTM (OMG UML), Superstructure version 2.4*. Object Management Group (OMG), Jan. 2011.
6. Oracle. *BPEL Process Manager Developer's Guide*, http://download.oracle.com/docs/cd/E15523_01/integration.1111/e10224/bp_sensors.htm. Accessed: Jun. 2011.
7. Sybase. *Sybase CEP Implementation Methodology for Continuous Intelligence*, http://www.sybase.com.au/files/White_Papers/Sybase_CEP_Implementation_Methodology_wp.pdf. Accessed: Jun. 2011.
8. A. H. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment*. Springer, 2010.