Monitoring BPMN-Processes with Rules in a Distributed Environment

Lothar Hotz[1], Stephanie von Riegen[1], Lars Braubach[2], Alexander Pokahr[2], and Torsten Schwinghammer[3]

[1] HITeC e.V. c/o Fachbereich Informatik, Universität Hamburg, Germany {hotz, svriegen}@informatik.uni-hamburg.de
[2] VSIS, Fachbereich Informatik, Universität Hamburg, Germany {braubach, pokahr}@informatik.uni-hamburg.de
[3] Uniique AG, Hamburg, Germany
Torsten.Schwinghammer@UniiqueAG.com

Abstract. In this paper, we demonstrate an application of rules in a business process scenario. As business processes, we consider data-intensive applications which need to move huge data files from server to server. By using the Business Process Model and Notation (BPMN) in our application, we enable clearly and hierarchically represented business processes. Such modeled processes can automatically be executed in a distributed environment with the Jadex open source middleware. Furthermore, the process execution is monitored with declarative rules, also implemented with Jadex. The demonstration shows the start of BPMN-modeled processes and their execution monitoring through logs and rules.

Keywords: Distributed systems, BPMN, rule-based systems, monitoring

1 Introduction

In business intelligence scenarios a huge amount of continuously growing data files has to be processed in distributed environments. Examples are log file, database, and campaign management as they are common in banking or telecommunication organizations. For such tasks, organizations use data integration approaches. However, (Friedman et al., 2008) points out that the "...commitment for implementing and supporting custom-coded or semi-manual data integration approaches is no longer reasonable" caused by the need for cost control. Consequently, organizations do already use specific data integration applications, however, for controlling data flows on distributed systems, manual activities or individual processes are still needed.

The basic principle in such applications consists of an extraction of data files from an operative system (like a web server), transformation of the data (on a staging server), and storing it in an analytical platform (like a data warehouse), see Figure 1. Data integration tools already handle diverse data file formats, however, they blank out that organizational data primary exist as decentralized, distributed data files. In our approach, we enable a declarative representation of business processes with the Business Process Model and Notation (BPMN) (OMG, 2006). With this notation, a user models business

processes on the basis of a predefined application component library. Those components implement basic tasks needed for file manipulation or similar activities. Such modeled processes can directly (i.e. without further coding) be executed in a distributed environment, such that subprocesses or tasks run on different servers.

In this paper, we focus on the monitoring of business process execution. This monitoring task has the goal to identify interesting occurrences during process execution. Such occurrences can be normal process execution as well as failing executions like not finished processes or exceeded average runtime of processes. For this process observation tasks, we apply a rule-based approach (see (Barringer et al., 2010) for a similar approach).

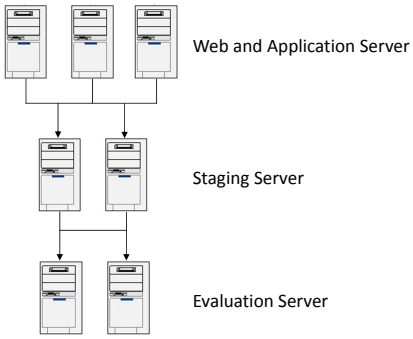

Fig. 1: Example Architecture

We are currently developing a system for automating business processes for gaining flexibility in process arrangement, quality improvements of data processing, and cost savings. The system consists of distributed, autonomously acting components, which are centrally controlled. Besides distributed quality assurance processes and integrative information life cycle strategy, the system has a process modeling component based on the BPMN (see Section 2) and a monitoring component (see Section 4), which are presented in this paper. We use the Jadex system[4] as a infrastructure supporting distributed agents and rules (see Section 3). In Section 5 and with a video[5], we demonstrate the processing of agents and rules with a data staging example.

2 BPMN Example

The de facto standard to graphically model applicable business processes is the Business Process Model and Notation (BPMN) (OMG, 2006). This notation is suited to formulate e.g. organizational structures or data models with elements such as events, activities, gateways, data objects, and transactions arranged in a pool and lanes. We use BPMN in the business intelligence context for distributed management of processes and data.

[4] http://jadex-agents.informatik.uni-hamburg.de
[5] http://monitoringrules.dyndns.org/

In the following, we introduce a simple example use case. Figure 1 depicts an exemplary server setting where the first stage of server outputs large amounts of data (for example customer informations collected by a publicity campaign) which will be processed by the staging server. The staging server filters the input data according filters like e.g. correct addresses. The quality analysis of data provided by the evaluation server brings the setting to a close.

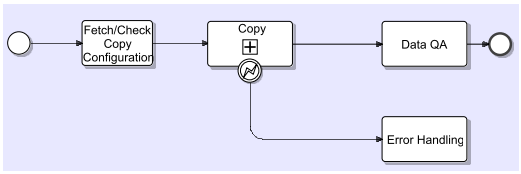

Fig. 2: BPMN model for data staging example

The notational model of the data processing is described in Figure 2. Before processing the collected data, the configuration data for the upcoming copy process has to be fetched and checked, possible configuration contents might be access credentials. The copy task is a collapsed subprocess (readily identifiable by the plus), in case of errors within this subprocess the error event leads to the error handling task. The expanded copy process is shown in Figure 3. After the data processing, the analyzing of quality (QA) step follows.

The expanded copy subprocess contains the following tasks: First the connection to a specific server has to be established, and before copying the data the existence is checked. Since of some data only one version is allowed to exist, a delete task is integrated via the exclusive gateway. Each task is bonded with a error catching event leading to the error handling task. Because of server-side connection problems, some errors might be corrected by a simple retry after a couple of minutes, see the timer event in Figure 3.

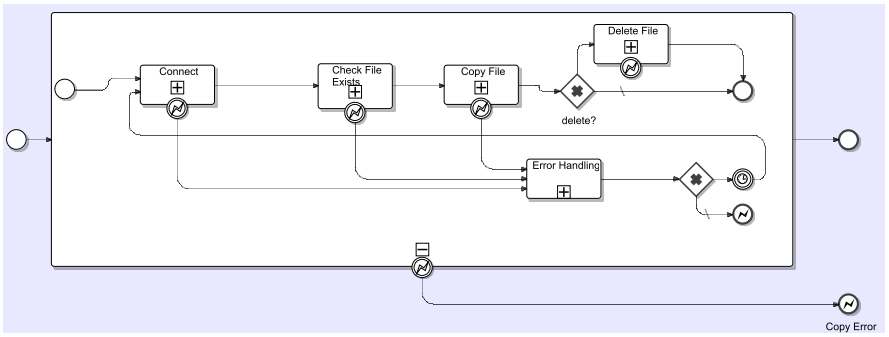

Fig. 3: Inner copy model of staging example

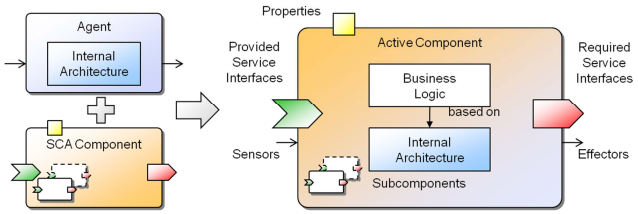

Fig. 4: Active Component

3 Jadex

The implementation platform used in this project is the open source platform Jadex (Braubach and Pokahr, 2011), which is a middleware for distributed systems. Jadex uses the new notion of active components for programming distributed systems. An active component or instance represents a unification of the component with agent and service concepts (cf. Fig. 4). The basic idea consists in having autonomously acting and loosely coupled entities in the sense of the actor model, i.e. communication among entities should be asynchronous and data objects should belong only to one actor to avoid data inconsistencies caused by concurrent accesses. This world model is combined with component and service ideas to strengthen the software engineering mechanisms that can be used to build software systems. The beneficial combination of component and service approaches has recently been put forward by the service component architecture (SCA) (Marino and Rowley, 2009), which introduces a component based architecture description framework for service oriented applications. The SCA component model defines a component with provided and required service interfaces to clearly state offered functionalities and dependencies. Furthermore, such interfaces allow for hierarchical (de)composition of complex software systems and foster reuse of software by building composites from readily available components. An active component further combines these SCA component concepts with agent characteristics, mainly it adds an internal architecture to a component. This internal architecture determines the type of an active component and the way its behavior has to be programmed. In this way very different kinds of active components such as BPMN workflows and even cognitive belief-desire-intention (Rao and Georgeff, 1995) agents can interact seamlessly because they share the same black-box view of each other.

3.1 The Runtime Platform

The Jadex runtime environment consists of the component container called platform and an additional tool set mainly used for administration, debugging and testing purposes. The Jadex platform facilitates the development of distributed systems in the following ways:

– Distribution transparency is established between components, i.e. a component can invoke a service of another component without having to know if this component is local or remote.

– An overlay network is automatically built by platform awareness. This means that Jadex platforms automatically find each other in local as well as distributed networks employing multiple different techniques such as IP broadcasts for local detection. In this way services of new remote platforms can be found and used as soon as a new platform has been discovered. Of course, the network building can be customized or disabled for specific customer settings.
– Platform security is assured. On the one hand Jadex introduces security mechanisms to protected the privacy of user data and services by separating awareness from communication means, i.e. platforms may detect each other but communication between them is restricted with respect to security settings. On the other hand application services can be declaratively equipped with security features so that authentication, confidentiality and integrity of communication partners is ensured. This is achieved by relying on established security protocols such as SSL.

3.2 Workflows and Rule Support

BPMN workflow support for Jadex consists of a visual editor based on the open source eclipse stp editor and the workflow engine that is able to execute modeled workflows. The editor mainly extends the stp version with new input fields for annotating implementation data that is necessary for executing the workflow. Such modeled workflows can be directly loaded and executed within Jadex using the BPMN kernel, which enacts each workflow as a separate active component instance. A workflow can spawn new subworkflows either locally or on remote platforms and monitor their execution. Furthermore, as workflows are components, they can invoke services of other workflows or components via their provided service interfaces. Rule support is based on a typical forward chaining rule engine called Jadex Rules that is similar to JESS and Drools, but targeted towards a very lightweight engine solution that can be integrated with other application parts. One reason for such a lightweight solution was the requirement to be able to execute a rule engine as part of an active component, i.e. due to the actual number of such components many rule engines have to run concurrently.

4 Monitoring

The duty of the monitoring component is to observe process execution and signal successful or failure execution. This monitoring can depend on application specific attributes like duration of process execution time or transfered file size. As common for a rule-based approach, working memory elements based on templates and rules can be used for representing the involved data and knowledge. Templates describe via fields structured data objects. Rules consist of a condition and action part. If some working memory elements fulfill the condition part of a rule, the rule system executes its action part.

In our application, while tasks and processes are executed, *logs* are created within application components. We differentiate between effective and failure logs. The effective logs are grouped by BPMN process, micro agent, rule, and task start and end logs. Every time an agent, BPMN process, or a task is started or will terminate shortly after,

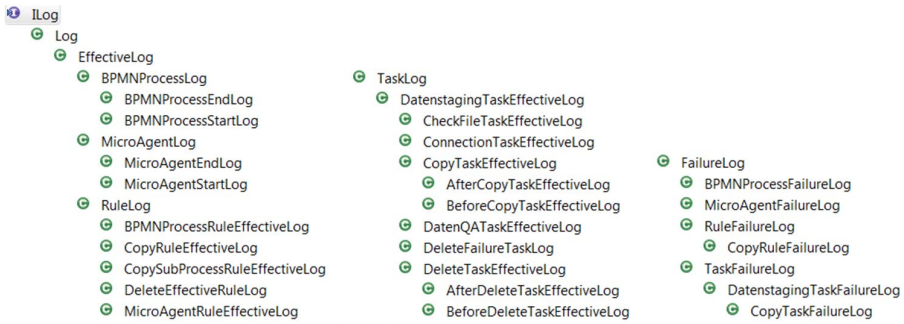

Fig. 5: Template hierarchy

a log will be created. In case of a failure within a task, rule, agent, or BPMN process a failure log is created. The basic layout of a log consists of the creation time, id, type, message, and parent id, but each log subtype extends this layout. For an overview of the currently used log types (implemented as Java classes), see Figure 5.

Working memory elements represent logs in the rule system. Consequently, in the condition part of a rule, types of logs and their fields can be tested. If a certain combination of logs was created, i.e. corresponding components were executed, a rule can fire and, thus, signal its execution with a further log. The rule depicted in Figure 6 creates a log of type *CopyRuleEffectiveLog* if two logs of type *BeforeCopyTaskEffectiveLog* and *AfterCopyTaskEffectiveLog* of the same process exist. Logs created through rule firing can be used in further rules for hierarchically testing log structures.

Thus, the application component implementor can model logs of a related type and rules describing certain effective or failure situations of an application component.

```
(defrule CopyTaskRuleEffective
;; Matches log-objects of type BeforeCopyTaskEffectiveLog
;; which deal with files of size not equal 0.
  ?ctlog1 <- (BeforeCopyTaskEffectiveLog
               (taskStartTime ?tst)
               (sourceFilesize ?sfs)
               (processID ?pid1)
               (test(!= ?sfs 0)))
;; Matches log objects of type AfterCopyTaskEffectiveLog
;; which deal with of size not equal 0 and has the same
;; processID as above.
  ?ctlog2 <- (AfterCopyTaskEffectiveLog
               (processID ?pid1)
               (targetFilesize ?tfs)
               (taskEndTime ?tet))
;; Task start time must be less task end time
  (test (< ?tst ?tet))
  (test (!= ?tfs 0))
=>
;; Creation of a combined working memory element representing
;; the firing of the rule.
  (assert (CopyRuleEffectiveLog (logs ?ctlog1 ?ctlog2))))
```

Fig. 6: Example for a rule, written in CLIPS syntax, monitoring the copy task.

The monitoring component itself is implemented as an agent which continuously receives logs from the executing application agents (see Figure 7). This happens via so called *LocalMonitoringRepositories* and one central *MonitoringRepository* that store

the logs for later use. Thus, the monitoring component observes the execution of distributed acting agents in a central way. It further combines the results of the agents' activities through firing rules.

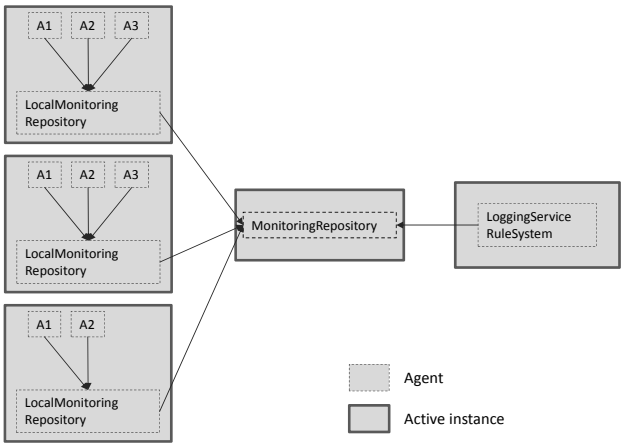

Fig. 7: Collecting logs from distributed agents running on different active instances

The monitoring component is part of a distributed business process execution application, which is currently under development. The application is implemented with JAVA and the extension Jadex for distributed agents.

5 Demonstrator

In the demonstrating example, we show the execution of an BPMN-modeled process for file transfer. For this task, following steps are executed:

- Start of the *Logging Service* agent that initializes the rule engine and waits for incoming logs.
- The specific working memory element *Clock* is initially created for representing the current time and, thus, enabling time-dependent rule firing.
- The BPMN-process is started by the user. Internally, new agents are created which follow the process model and execute the related basic component implementation. During the process execution specific logs are created like *BPMNProcessStartLog* and *BeforeCopyEffectiveLog*.
- Created logs and activated rules can be examined in a rule monitor window.
- Each fired rule creates new logs like the *CopyRuleEffectiveLog*.
- In a further run-through of the demonstrator another rule fires indicating that a copy task needs too much time to be processed.

6 Discussion and Summary

Other data integration tools already handle diverse data file formats, however, they blank out that organizational data primary exist as decentralized, distributed data files. User

of such systems are forced to manually move the data files to appropriate places or to develop scripts and programs that move them around in distributed systems. To the best of our knowledge, no other business intelligence software is focusing on the process execution monitoring via rules. Hereby, processes report their execution via logs (e.g. start and end logs) and rules observe those for identifying interesting monitoring occurrences. This rule-based approach has following advantages. First, we decouple the execution of the actual processes from their monitoring, i.e. we separate application logic from monitoring logic. If new interesting occurrences shall be recognized during execution the process component implementation has not to be changed but only rules have to be added. Thus, maintenance shall be simplified. Furthermore, by using rules, we allow a declarative representation of such interesting situations which can be modeled by domain experts, in the ideal case, e.g. when domain specific languages are introduced for rule modeling (see (Laun, 2011)). Such models (rules) can reflect on single processes as well as combinations of different processes or subprocesses. Similarly, results of rule firing can be aggregated through rule chaining. Thus, beside the typical procedural representation of process' behavior in BPMN diagrams, rules provide a declarative representation of the expected outcome of process execution. When rules are fulfilled, such informations can again be stored in repositories or communicated to user interfaces that present actual states of process execution (e.g. in a business process browser). The data-driven character of rule-based approaches enables a direct reaction and evaluation of current situations, like daemons who react actively on data occurrences. Contrarily, a database approach would need to actively apply queries on a database.

In this demonstration paper, we present a combination of process modeling based on BPMN, process execution in a distributed environment, and process execution monitoring with rules. The demonstrator shows how these technologies can successfully be combined to monitor process execution in a distributed environment.

References

Barringer, H., Rydeheard, D. E., and Havelund, K. (2010). Rule Systems for Run-time Monitoring: from Eagle to RuleR. *J. Log. Comput.*, 20(3):675–706.

Braubach, L. and Pokahr, A. (2011). Addressing Challenges of Distributed Systems Using Active Components. In Brazier, F., Nieuwenhuis, K., Pavlin, G., Warnier, M., and Badica, C., editors, *Intelligent Distributed Computing V - Proceedings of the 5th International Symposium on Intelligent Distributed Computing (IDC 2011)*, pages 141–151. Springer.

Friedman, T., Beyer, Mark, A., and Bitterer, A. (2008). Magic Quadrant for Data Integration Tools. Technical report, Gartner.

Laun, W. (2011). Domain Specific Languages: Notation for Experts. In *International Conference on Reasoning Technologies (Rules Fest)*, San Francisco.

Marino, J. and Rowley, M. (2009). *Understanding SCA (Service Component Architecture)*. Addison-Wesley Professional, 1st edition.

OMG (2006). *Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification.*

Rao, A. and Georgeff, M. (1995). BDI Agents: from Theory to Practice. In Lesser, V., editor, *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS 1995)*, pages 312–319. MIT Press.