Brauer, P. C. and Hasselbring, W. (2012) Capturing provenance information with a workflow monitoring extension for the Kieker framework [Paper] In: The Third International Workshop on the role of Semantic Web in Provenance Management , 28.5.2012, Heraklion, Kreta.

Capturing provenance information with a workflow monitoring extension for the Kieker framework

Peer C. Brauer
Wilhelm Hasselbring

Software Engineering Group, University of Kiel, Christian-Albrechts-Platz 4, 24118 Kiel

Abstract: Workflow technologies are getting more and more attention in daily business life. So why are these system not used for science? One reason why this is not feasible is, because there is a lack of provenance capturing features in business workflow engines. In this paper we present an approach to enhance business workflow engines with advanced monitoring features for capturing provenance information. We will discuss our approach with the help of the Apache ODE workflow engine[1], for which we will provide an exemplary provenance monitoring solution.

1 Introduction

The way research is done is changing rapidly. More and more experiments are shifted away from the laboratory bench to computer based simulations, even classic laboratory work moves to virtual working environments [Sch10]. Different scientific workflow engines like Kepler[2], Trident[3] or Taverna[4] have been developed to support scientists during their daily work. These tools are specially designed to meet all the requirements of a scientific working environment. Compared to business workflow engines they do not only have benefits, but also some disadvantages. One is, that they are developed by a much smaller community. Because of this, the software environment and tool landscape built around these workflow engines is much smaller than for commercial or open source business workflow systems. So why not also use these systems for science and benefit from all the tools, which were developed by the community or companies. One of the answers to this question is, that most business workflow engines lack of provenance capturing mechanisms.

This paper presents an approach for adding provenance capturing functionality to normal business workflow engines. This is done by creating an easily to configure and adaptable monitoring framework for business workflow engines. The approach is build upon the Kieker monitoring framework and the SCOPE DSL. It is tested within the Apache ODE workflow engine.

[1] http://ode.apache.org
[2] https://kepler-project.org/
[3] http://research.microsoft.com/en-us/collaboration/tools/trident.aspx
[4] http://www.taverna.org.uk/

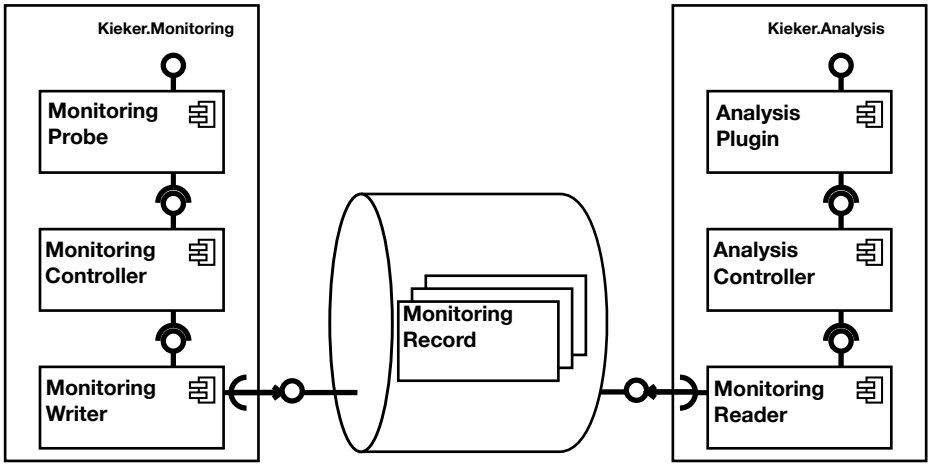

Figure 1: Kieker architecture overview

Kieker [vHWH12] is a modular monitoring framework, which can be easily extended by user created components. The development of the framework started in 2006. Since then, it has evolved from a small tool for monitoring the response times of Java applications to a powerful monitoring and analysis tool. Currently, the standard implementation provides modules for monitoring and analyzing not only the runtime behavior of software systems, but also it's inner structure. Therefore it includes different sorts of monitoring probes and analysis components.
As figure 1 depicts, Kieker is structured into different parts. The main components are Kieker.Monitoring and Kieker.Analysis. In the Kieker.Monitoring component reside the probes, which are responsible for collecting the monitoring data. Kieker already provides different probes for measuring values like CPU utilization, memory usage and other system characteristics. The collected monitoring data is then passed to the Monitoring Controller, which writes it,using the Monitoring Writer component, to a datastore.
The analysis of the monitoring data is performed within the Kieker.Analysis component. The monitoring data is read from the datastore by the Monitoring Reader and passed to the Analysis Controller. The Analysis Controller can start different Analysis Plugins containing chains of analysis filters, which interpret the collected monitoring data and provide different sorts of visualizations. In a later version, Kieker will provide a new feature called the Kieker.AnalysisGUI.[5] This feature will allow users to compose structures of filters and visualizations graphically.

SCOPE is a running research project in the context of space based multicore programming [GH11]. The goal of this project is to alleviate multithreaded Java programming. The programmer should not have to deal with the typical problems resulting from accessing concurrent data structures. SCOPE consists of two different parts. The first part is the SCOPE coordination model with focus on the choreography of the accesses to the space and the orchestration of the single processes. The other part is the SCOPE DSL. This

[5] You can trace the development stage at the project site (http://kieker-monitoring.de)

component focuses on separating the description of the collaboration from the process definition.

2 Monitoring workflows for collecting provenance data

2.1 How to describe what (provenance-) information should be logged

Since monitoring always has a negative impact on the performance of a computing system [vHRH+09, 11-14], it is essential to mark those areas of a process for which monitoring should be enabled. In the context of the Kieker.WorkflowMonitor we utilize the BPEL extension mechanism[6] to weave monitoring information directly into a BPEL process definition. We have implemented our own extensions, which can be added to all extensible elements of a BPEL process definition. These extensions indicate if monitoring should be active and what should be monitored.
To describe what to monitor, we developed so called profiles. A profile is an XML element describing which event to monitor and what type of information to collect. Listing 1 shows a sample profile.

Listing 1: a sample profile

```
1  <monitor:profile name="provenanceLight">
     <monitor:targetelement type="activity" >
       <monitor:eventtype>activityEnabledEvent</monitor:eventtype>
       <monitor:eventtype>activityDisabledEvent</monitor:eventtype>
     </monitor:targetelement>
6    ...
     <monitor:targetelement type="variable" >
       <monitor:eventtype>VariableModificationEvent</monitor:eventtype>
       <monitor:eventtype>VariableReadEvent</monitor:eventtype>
     </monitor:targetelement>
11   ...
     <monitor:event type="VariableModificationEvent">
       <monitor:fokus>varName</monitor:fokus>
       <monitor:fokus>newValue</monitor:fokus>
     </monitor:event>
16 </monitor:profile>
```

Using the BPEL extension we can now weave the information about which profiles to apply to which elements of the BPEL process directly into the process definition.
With these basics, it is quite easy to enable the monitoring framework to collect provenance information. All one have to do is to create a monitoring profile, which defines what monitoring information is also provenance information and weave the monitoring instructions into the process definition using the extension mechanism.

[6] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

2.2 How to get the monitoring data

As mentioned before the Kieker.WorkflowMonitor is built upon the Kieker monitoring framework. This has the advantage that the proven and well tested infrastructure of Kieker can be used. Kieker.WorkflowMonitor extends the Kieker framework with a special workflow probe and some new analysis plugins.

The new probe is an entirely new developed Kieker probe. Because of the special requirements resulting from the domain of workflow monitoring, none of the Kieker probes developed so far could be reused. The new probe was designed to meet the following requirements

- minimal affect to the performance of the workflow system
- allow changes to the monitoring behavior during runtime
- monitoring of distributed workflow systems

We decided to build a probe, that consists of two different parts. The first part has to collect the monitoring data and is embedded into the workflow engine, in this context this part is called the probe. The second part contains the logic, converts the monitoring data to an workflow independent exchange format and passes it to the persistence layer. It is called the preprocessor.
Because of the differences between the single workflow engines it is not possible to provide a probe, which works fine for every workflow engine. Therefore we decided to provide an abstract probe containing all the logic needed for workflow monitoring and the communication between the probe and the preprocessor. In this way users of the workflow monitoring framework can write a monitoring probe for the monitoring framework of their choice. They only have to overwrite the `onEvent()`-method of the abstract probe and provide a configuration for the communication channel. The `onEvent()`-method is the method which should be called by the workflow engine, when an event occurs. How this call can be realized is different from workflow engine to workflow engine. In Apache ODE, the workflow engine on which this approach is tested, the probe can be listed as custom event listener by adding it to the `ode-axis2.properties` file and implementing the BpelEventListener.
Whenever a runtime event within ODE is created, it is passed by the probe to the preprocessor. The preprocessor again consists of four different parts, the communication channel receiver, which writes the raw event logs to a log-space, the log-space, the log filters and the persistence adapter called the log-writer. To alleviate the multithreaded programming, which is necessary for a performant implementation of the preprocessor, the complete preprocessor is modeled and generated using SCOPE.

2.3 Boosting performance with SCOPE

As mentioned earlier, SCOPE is a framework for alleviating the programming of multithreaded JAVA programs, based on a DSL, describing a BPMN compliant model of the system, and a tuple space, for coordinating the access on shared resources. Listing 2 shows a short extract of the coordination model definition of the preprocessor.

Listing 2: SCOPE model

```
definitions org.scope.target.gen.workflowMonitoring.processes
{ ...

    collaboration WorkflowProbe {
        space LogSpace
        client WorkflowProbeClient : WorkflowProbeProc
    }

    process WorkflowProbeProc attends WorkflowProbe {
        Configuration config init;
        publish config to WorkflowProbe.LogSpace;
        parallel {
            multi-instance (config.maxNumberOfReceivers) call
                LogReceiver;
            multi-instance (config.maxNumberOfPreprocessors)
                call LogWorker;
            call LogWriter;
        }
    }

    process LogWorker attends WorkflowProbe {...}

    process LogWriter attends WorkflowProbe {...}

    process LogReceiver attends WorkflowProbe {...}
}
```

Beside the tuple space, the preprocessor consists of three different parts. The LogReceiver receives the monitoring events, which were sent by the probe, asynchronously and writes them to the tuple space. When the workflow monitoring framework is used for monitoring different instances of workflow engines or when there is a need to speed up the communication channel, the number of receivers can be increased by altering the `config.maxNumberOfReceivers`-value in the configuration file.
The LogWorker threads are responsible for filtering the events monitored. This is necessary, because the probe sends all events it registers to the space. If the probe started to filter out the events which should not be monitored, this would decrease the performance of the workflow engine en bloc. Instead the filtering is done in the preprocessor. The preprocessor interprets the profiles mentioned earlier and applies the monitoring instructions to the events listed in the tuple space. In this way, all monitoring information which should not be persistent, are rejected. After the filtering process all remaining event logs are transformed into a generic exchange data format for monitoring information. This exchange format is based on a ontology defined in the Web Ontology Language (OWL).[7]

[7] http://www.w3.org/TR/owl2-overview/

Finally, the LogWriter, which is derived from the Kieker Monitoring Writer, is responsible for writing the acquired and filtered monitoring information to a persistent data store.

3 Conclusion and future work

In this paper we presented an approach for monitoring business workflow systems based on the Kieker framework. To allow Kieker to monitor business workflow engines, it is extended with a new probe and new monitoring data analysis components . The new probe can easily be used to implement own probes for different workflow engines. By defining monitoring profiles, which are interpreted by the preprocessor and used for filtering the events monitored by the probe, the monitoring behavior of the framework can be controlled.
A future task will be to create graphical representations for the monitoring results, which will be embedded into the new Kieker analysis GUI currently being developed. Therefore it is necessary to design filters and visualizations for workflow monitoring, which can be used inside the framework. Another outstanding task is the definition of basic monitoring profiles, describing the monitoring behavior of the probe for some basic use cases like performance monitoring or business activity monitoring. To improve the interoperability with other provenance capturing systems and analysis tools, data export functions to the Open Provenance Model [MCF+10] should be added.

References

[GH11] Stefan Gudenkauf und Wilhelm Hasselbring. *Space-Based Multi-Core Programming in Java*, Seiten 41–50. The Associacion for Computing Machinery, Inc., 2011.

[MCF+10] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan und Jan Van den Bussche. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, July 2010.

[Sch10] Matthias Schulze. Virtuelle Forschungsumgebungen und Forschungsdaten für Lehre und Forschung: Informationsinfrastrukturen für die (Natur-)Wissenschaften. In *Ockenfeld, Marlies (Hrsg.): Semantic Web and Linked Data : Elemente zuknftiger Informationsinfrastrukturen; Proceedings der 1. DGI-Konferenz, 62. Jahrestagung der DGI, Frankfurt am Main, 7.-9. Oktober 2010. Frankfurt am Main, 2010 (Tagungen der DGI 14), S. 165-176*. Universitt Stuttgart, 2010.

[vHRH+09] André van Hoorn, Matthias Rohr, Wilhelm Hasselbring, Jan Waller, Jens Ehlers, Sören Frey und Dennis Kieselhorst. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Bericht TR-0921, Department of Computer Science, University of Kiel, Germany, November 2009.

[vHWH12] André van Hoorn, Jan Waller und Wilhelm Hasselbring. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. ACM, April 2012. Invited tool demo paper. To appear.