Preprint of: Peer Brauer, Florian Fittkau, Wilhelm Hasselbring: The Aspect-Oriented Architecture of the CAPS Framework for Capturing, Analyzing and Archiving Provenance Data. In: Proceedings of the 5th International Provenance and Annotation Workshop (IPAW 2014), Cologne, Germany, Springer-Verlag LNCS, June 2014.

The Aspect-Oriented Architecture of the CAPS Framework for Capturing, Analyzing and Archiving Provenance Data

Peer Brauer,[1] Florian Fittkau[1] and Wilhelm Hasselbring[1]

[1] Software Engineering Group, Kiel University, Kiel, Germany
{pcb,ffi,wha}@informatik.uni-kiel.de

With aspect-oriented programming techniques, modularity may be achieved via separating cross-cutting concerns. Data provenance can be considered as a cross-cutting concern: code for collecting provenance data is usually scattered across various places in a software system. Aspect-oriented programming allows to seamlessly integrate cross-cutting concerns into existing software applications without interference with the original system.

Following this approach, CAPS[1] is a framework to weave provenance-capturing mechanisms into existing Java applications, which are not yet provenance aware. The CAPS framework employs AspectJ [5],[2] the Kieker framework [7,4],[3] the Java Management Extensions JMX,[4] and some Java security mechanisms to automatically collect the provenance information. Woven inside the application as a minimal-invasive integration of the provenance capturing mechanisms, CAPS monitors the execution of the software. Whenever a data set is processed, CAPS creates the corresponding provenance graph entry. The graph itself is stored in an integrated provenance archive build on top of the Neo4j graph database.[5] CAPS is implemented and evaluated in the context of the PubFlow workflow system for semi-automatic research data publication [2]. In particular, workflow-generated provenance data is automatically gathered via CAPS, without mixing program logic with provenance mechanisms.

For deployment, CAPS provides a GWT-based web interfaces,[6] which allows the user to upload his own scientific Java applications to the CAPS runtime environment. While uploading the application, the user has to provide basic information about the application and its runtime environment. These include:

- the deployment type of the application (e.g., web based, Java archive),
- virtual machine parameters,
- application parameters and
- the URL of an existing CAPS Provenance Archive instance in case of standalone applications.

[1] CAPS stands for **C**apturing and **A**rchiving **P**rovenance in **S**cientific workflows
[2] http://eclipse.org/aspectj/
[3] http://kieker-monitoring.net/
[4] http://docs.oracle.com/javase/tutorial/jmx/
[5] http://www.neo4j.org/
[6] http://www.gwtproject.org/

Based on the provided information, CAPS suggests a so-called application profiles for the application to be deployed. A profile contains a predefined selection of aspects and Kieker monitoring probes, that are applicable to the type of the given application. CAPS also provides profiles for Java-based workflow systems such as jBPM.[7] The user can refine the suggested profile or switch to another profile that collects more detailed information profile.

After selection of the profile to be applied to the application, CAPS creates a runtime configuration based on the provided information. After the creation of the profile, the user may check the configuration via a profiling run.

If the user chooses to initiate a profiling run, the system starts the application and displays the provenance information, captured by CAPS. This provides the user the opportunity to check, whether all relevant aspects of the system are under surveillance, and whether the monitoring level should be increased or decreased. The user can repeat this process to optimize the provenance trace produced by CAPS.

CAPS uses the Java sandbox security mechanism to intercept I/O and network calls.[8] We employ these components by weaving our monitoring probes directly into those methods that are responsible for checking the applications' calls against the JVM security constrains. CAPS also alters the configuration of the JVM for the client application which always activates the sandbox, whenever the application starts. It also obtains additional basic runtime information about the client application by querying the JMX interface.

Next, the user has to decide, whether the application should be exported as a standalone application, such that it can be used without CAPS, or whether the application should be added to the CAPS application library. For standalone applications, CAPS creates a so-called CAPS connector and embeds it into the application. The connector is responsible for connecting the application to the CAPS server, so the provenance data created by the application can be analyzed and archived.

To extract the provenance information from the collected monitoring data, CAPS utilizes the existing data analysis functionality of the Kieker framework, i.e. the analysis framework and the Kieker WebGUI [3].

CAPS provides specific Kieker filters, that can be used to filter the provenance data from the stream of monitoring records. These filters is described in [1]. CAPS comes with predefined analysis components, and offers the user to create her own analysis components. Predefined analyses are, for example, available for creating the PROV-O[9] provenance graph or for reconstructing workflows in scientific workflow environments.

To store the provenance information collected by the framework, CAPS uses an integrated provenance archive. The archive is built on top of the Eclipse

[7] http://www.jboss.org/jbpm

[8] http://docs.oracle.com/javase/7/docs/technotes/guides/security/spec/security-spec.doc1.html

[9] http://www.w3.org/TR/prov-o/

Modeling Framework Project (EMF),[10] the Google Web Toolkit (GWT)[11] the PubFlow Graphframework,[12] and Neo4j. It was a result of the W3C call for implementations of the PROV-O data model.[13] The provenance archive is developed based on an extended version of the PROV-DM [6], implemented with the Eclipse Modeling Framework. We made small additions to the PROV-DM model, such that we can store some additional information, like execution time stamps and user roles. However, we keep our model compatible to the original W3C PROV-DM. As persistence layer for our provenance archive we chose a Neo4j graph database. This offers the advantage of benefiting from the specific graph algorithms provided by the database engine. To store our EMF model in the graph database we are currently building a new persistence layer based on neo4emf,[14] a framework that allows mapping an EMF model to a Neo4j database.

References

1. Brauer, P.C., Hasselbring, W.: Capturing provenance information with a workflow monitoring extension for the Kieker framework. In: Proceedings of the 3rd International Workshop on Semantic Web in Provenance Management. CEUR-WS (May 2012), http://eprints.uni-kiel.de/19636/
2. Brauer, P.C., Hasselbring, W.: PubFlow: a scientific data publication framework for marine science. In: Proceedings of the International Conference on Marine Data and Information Systems (IMDIS 2013). vol. 54, pp. 29–31 (September 2013), http://eprints.uni-kiel.de/22399/
3. Ehmke, N.C.: Everything in sight: Kieker's WebGUI in action. In: Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days 2013. pp. 11–19. CEUR-WS (Nov 2013), http://eprints.uni-kiel.de/22528/
4. van Hoorn, A., Waller, J., Hasselbring, W.: Kieker: A framework for application performance monitoring and dynamic software analysis. In: Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012). pp. 247–248. ACM (April 2012), http://eprints.uni-kiel.de/14418/
5. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of AspectJ. In: ECOOP 2001 Object-Oriented Programming, pp. 327–354. Springer (2001)
6. Moreau, L., Missier, P.: PROV-DM: The prov data model. Tech. rep., World Wide Web Consortium (2013)
7. Rohr, M., van Hoorn, A., Matevska, J., Sommer, N., Stoever, L., Giesecke, S., Hasselbring, W.: Kieker: Continuous monitoring and on demand visualization of Java software behavior. In: Proceedings of the IASTED International Conference on Software Engineering 2008 (SE'08). pp. 80–85 (Feb 2008)

[10] http://www.eclipse.org/modeling/emf/
[11] http://www.gwtproject.org/
[12] http://www.pubflow.uni-kiel.de/en/the-framework/the-graphframework
[13] http://www.w3.org/TR/2013/NOTE-prov-implementations-20130430/
[14] http://neo4emf.com/