MooseJEE: A Moose Extension to Enable the assessment of JEAs

Fabrizio Perin

Software Composition Group
University of Bern, Switzerland
http://scg.unibe.ch

***Abstract*—Java Enterprise Applications (JEAs) are large systems that integrate multiple technologies and programming languages. With the purpose to support the analysis of JEAs we have developed MooseJEE an extension of the *Moose* environment capable to model the typical elements of JEAs.**

Keywords: Java enterprise, Tool, Reverse engineering.

I. Introduction

Java Enterprise Applications (JEAs) are large systems written in multiple languages and integrating various technologies. Information concerning the application's elements and their relationships is spread over various source files. This causes developers to lose the overview of the system and hides inconsistencies in the code.

To address these problems we have developed MooseJEE an extension to the *Moose*[1] environment that supports the inspection and analysis of JEAs. Specifically, MooseJEE provides importers to collect information from various sources, a number of dedicated software visualizations, and a code browser to explore the issues detected by MooseJEE directly in the application source code. The MooseJEE importers are necessary to collect additional information about JEAs not present in the Java source code (*e.g.*, information from XML configuration files or SQL scripts). The MooseJEE visualizations tackle the problems of identifying the transaction scope of applications and exposing architectural inconsistencies [1]. While software visualizations are useful to obtain an overview of the system, it is also important to maintain the relations with the code under analysis. This is why we developed the MooseJEE code browser that allow the inspection of the source code directly into the Moose environment.

We introduce MooseJEE and its features in section II, we present the MooseJEE code browser in section III and we present a usage scenario in section IV.

II. MooseJEE

MooseJEE is an extension of Moose[2], a language-independent environment for reverse- and re-engineering complex software systems. Moose internally uses *FAMIX* [3], a language-independent meta model, to describe the static structure of object-oriented software systems. FAMIX was not able to express completely the complexity and the heterogenous nature of JEAs. This is why we have extended the FAMIX meta model with representations of typical elements of JEAs. These extensions are needed to represent Enterprise Java Beans (EJBs), the usage of relational databases, and architectural layers typically found in JEAs [4].

Armed with a suitable meta model to describe JEAs, we extend the standard Moose environment with functions to import information from multiple sources and to inspect this information. All these functions are accessible through the standard Moose Panel.

MooseJEE adds two importers to the standard Moose panel menu to load into the Moose environment the model of a JEA. It has been necessary to create different importers to take into account the significant differences between EJB version 2.1 [5] and version 3.0 [6]. To build an actual model we use inFusion (the successor of iPlasma [7]) to parse Java files and build a basic model, and then we separately parse other information sources (*e.g.*, the *ejb–jar* XML file) and integrate the new data into the model. Triggering one of the two importers causes a wizard to appear and prompt the user for all the information required to set up the model.

The following operations are performed by the importers:

- Common operations
  1) Create a model from an MSE file.
  2) Annotate the methods, the classes and the packages of the model indicating if they are part of a Transaction Scope or an Unsafe Path [1].
  3) (optional) Open a code browser or a software visualization on the imported project.
- EJB 2.1
  1) Import information about EJBs from the *ejb-jar.xml* file.
- EJB 3.0
  1) Import information about EJBs from Java annotations.
  2) Import information about the relational database from SQL files.
  3) Describe the relations between the database elements and the software elements.

The functions previously described can also be manually triggered directly on a Moose model already imported using

[1]http://www.moosetechnology.org/

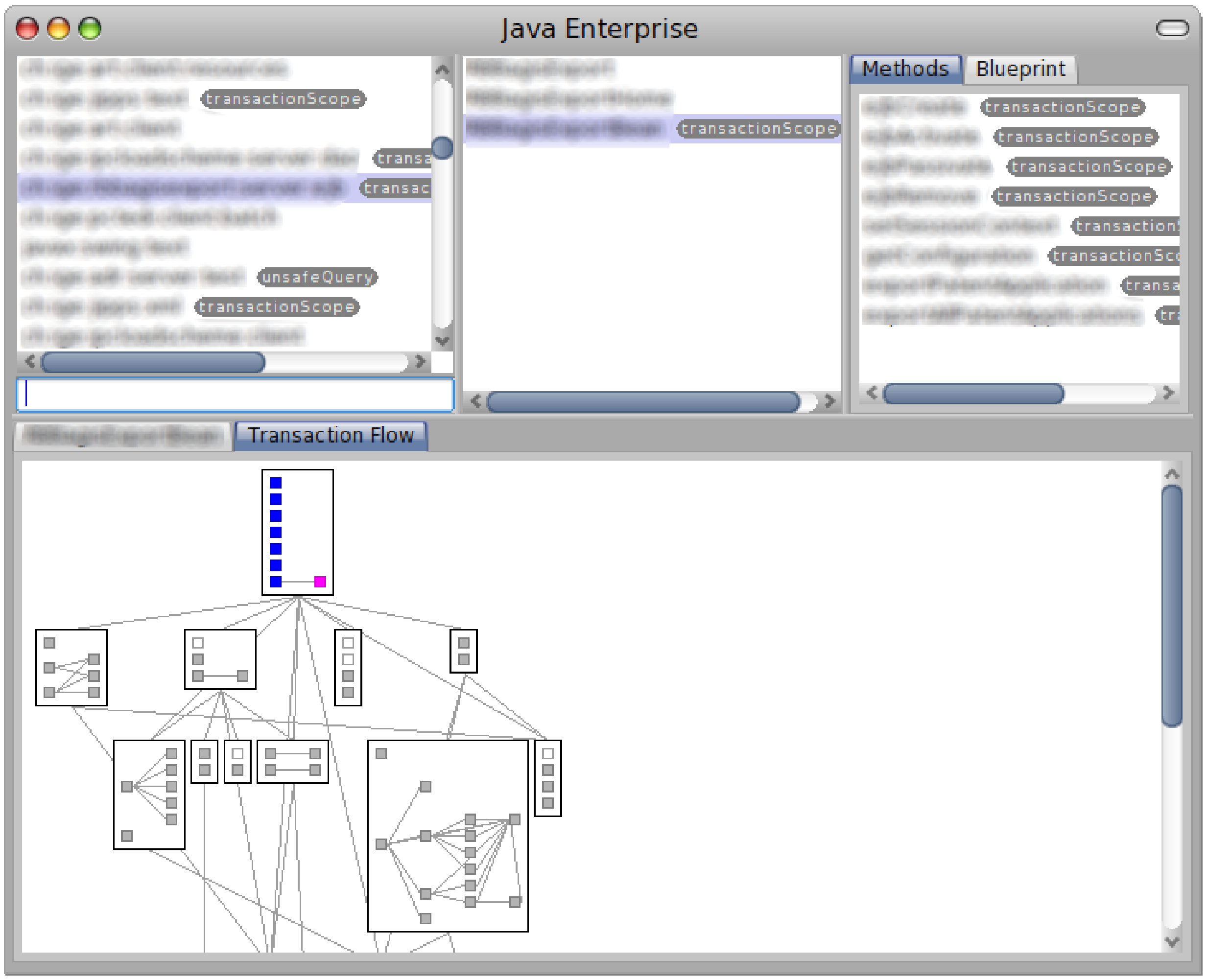


Fig. 1. Code browser for Java Enterprise applications (package, class and method names obscured due to NDA)

standard instruments in the Moose panel. These utilities have been grouped in a new menu entry called "JEAs Menu" of the standard context menu of Moose models.

The current implementation of MooseJEE contains software visualizations useful for the recovery and analysis of transaction scope in JEAs. These visualizations are accessible from a new entry called "JEAs Visualizations" of the context menu of any class group entry in the Moose Panel.

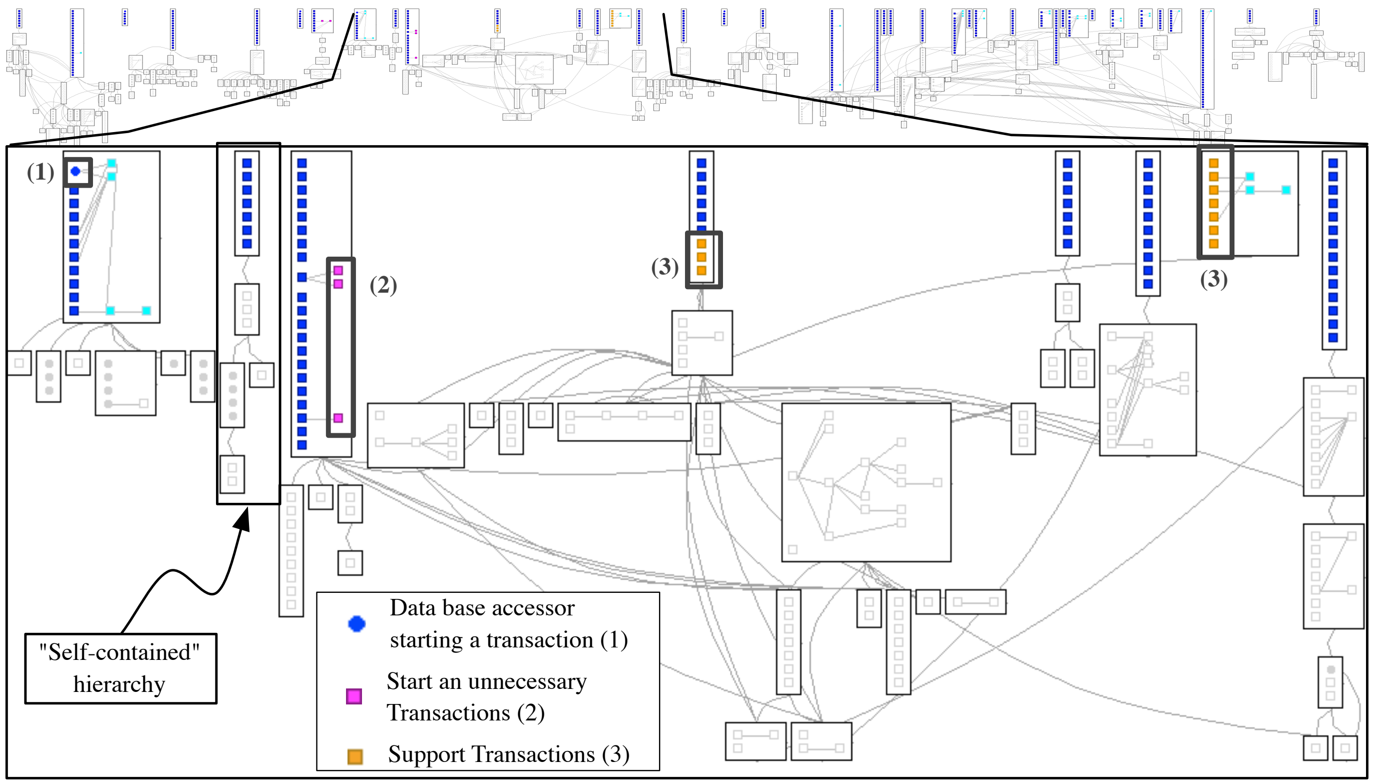
III. Code browser

Visualizations provide valuable instruments to obtain an overview of a system but the risk is to lose the contact with the source code. The code browser is therefore an important instrument in the process of software understanding to offer an immediate comparison between the visualizations and the source code. This is why we have developed a code browser that also incorporates two software visualization. The browser has been developed using Glamour [8], a framework dedicated to building browsers.

The code browser (Figure 1) is organized as follows: the first panel contains the list of the project's packages, the second contains the list of classes contained in the selected package, the third contains the list of methods of the selected class and the *class blueprint* [9]. In the panel below it is possible to see the source code of the selected class or method and the transaction flow visualization of the hierarchy containing the selected class. In the first three panels the elements that are part of a transaction scope or part of an unsafe path that access the database are highlighted with a gray label. An *unsafe path* is an invocation chain that starts from an entry point method that does not start an application transaction [1]. It is also possible to filter the elements by clicking on the gray label indicating a property, only the elements with that property will be shown. The code browser does not aim to substitute the development environment but just to provide an instrument to inspect the source code directly from Moose.

IV. Tool Demonstration

We will demonstrate how to use the MooseJEE extension to perform an inspection of the transaction scope consistency in JEAs. It is possible to download MooseJEE from `http://scg.unibe.ch/research/Moose-JEE`; this



(1)
(2)
(3)
(3)
"Self-contained" hierarchy
● Data base accessor starting a transaction (1)
■ Start an unnecessary Transactions (2)
■ Support Transactions (3)

Fig. 2. An excerpt from a Transaction Flow Visualization with some important elements highlighted

site also provides the installation instructions and a video of the following scenario.

The application that MooseJEE can analyze are: all the JEAs using EJBs 2.1 and EJBs 3.0. MooseJEE importers can collect information about the Java Beans from the Java annotations and from the *ejb-jar* XML configuration file. MooseJEE can recognized the accesses to a data base if the application uses the *Java.sql* package. If the application uses tools like Hibernate to deal with the persistency MooseJEE will not be able to identify the data base accesses. This limitation will be cover on the future steps of the work.

Before launching MooseJEE it is necessary to use *inFusion*[2] to generate an MSE file of the Java application that the user wants to analyze. *inFusion* is an integrated environment for performing code and architectural reviews of object-oriented and procedural software systems. With the MSE file of the application available it is possible to start using MooseJEE from the standard Moose Panel.

The scenario will deal with three software visualizations having the following purposes:

a) Transaction Flow: this visualization has been built to analyze the application transaction scope in JEAs, see Figure 2. The main purpose is to expose all the methods that can start a transaction when it is not strictly necessary ((2) in Figure 2) and to highlight the presence of invocation chains that support the usage of transaction but that do not start one by themselves ((3) in Figure 2). The "Transaction Flow" visualization is also useful to identify parts of code that are "self-contained" in the sense that their entry point and their implementation is not related to other elements of the application. On the other hand it is also possible to identify more complex hierarchies with multiple entry points sharing various classes. The identification of these structures may be useful to guide refactoring to make application services more independent.

b) Unsafe Query: thank to this visualization it is possible to identify the methods that are not involved in an application transaction but access the database. This inspection is important because helps to identify part of code managing with possible inconsistent data into the application and also highlights flows of execution that cannot be rollbacked.

c) Server Layers: this visualization exposes structural violations of the elements involved in a transaction. In particular it highlights all Session beans that are in the wrong layer and all invocations that jump a layer. Such violations of architectural constraints not only impact program comprehension, but they may be signs of more serious defects in the application code. *e.g.*, in Figure 2 the method marked with (1) is a method that access the data base but it is actually contained in a session bean. This fact imply that the Session Bean containing that method is considered part of the data layer [4] and not part of the Service Layer [4].

In Table I are listed the steps to perform an inspection of the transaction scope consistency in a JEAs.

[2]http://www.intooitus.com/inFusion-tryit.html

	Steps	Output	Comments
		Common Steps	
1	Create the MSE file of the application using inFusion	An MSE file	Refer to the inFusion web site for further information http://www.intooitus.com/inFusion-tryit.html
2	Select in the Moose Panel Menu the importer for EJBs 2.1	A wizard appears asking the user to insert various pieces of information	The wizard requires information like the location on the disk of the MSE file of the application and other similar requests
3	Fulfill the requests of the wizard's panels that will appear in sequence	At the end of the import process, appears in the Moose Panel appear the name of the model imported	A progress bar will appear to keep the user informed about the ongoing import process
4	Select the model in the Moose Panel	At the end of the computation a pane appears on the right showing some elements contained into the model	Refer to the Moose web site for more details about the Moose Panel http://www.moosetechnology.org/
		Transaction Flow	
5	Right click on the "All model classes" entry and select "Transaction Flow" under the "JEAs Visualizations" entry	A Transaction Flow visualization appears, see Figure 2	More details about the visualization in [1]
6	Right click on the Moose model contained in the Moose Panel and we select "Code Browser for JEA" contained into the "JEAs Menu" entry.	A code browser opens allowing the access to the source code of the application see Figure 1	
		Unsafe Query	
5	Right click on the "All model classes" entry and select "Unsafe Query" under the "JEAs Visualizations" entry	An Unsafe Query visualization appears [1]	More details about the visualization in [1]
6	Right click on the Moose model contained in the Moose Panel and we select "Code Browser for JEA" contained into the "JEAs Menu" entry.	A code browser opens allowing the access to the source code of the application	
		Server Layer	
5	Right click on the "All model classes" entry and select "Server Layer" under the "JEAs Visualizations" entry	A Server Layer visualization appears [1]	More details about the visualization in [1]

TABLE I
STEPS TO FOLLOW TO DISCOVER INCONSISTENCIES IN THE TRANSACTION SCOPE OF A JEA.

Acknowledgments - We gratefully acknowledge the financial support of the Hasler Foundation for the project "Enabling the evolution of J2EE applications through reverse engineering and quality assurance" (Project no. 2234, Oct. 2007 – Sept. 2010). We also gratefully acknowledge the contribution of our (anonymous) industrial partner for providing the case study used in this work. We thank Oscar Nierstrasz, David Röthlisberger, Jorge Ressia and Lukas Renggli for their comments on drafts of this paper.

REFERENCES

[1] F. Perin, T. Gîrba, and O. Nierstrasz, "Recovery and analysis of transaction scope from scattered information in java enterprise applications," in *Proceedings of International Conference on Software Maintenance 2010*, Sep. 2010, to appear.

[2] O. Nierstrasz, S. Ducasse, and T. Gîrba, "The story of Moose: an agile reengineering environment," in *Proceedings of ESEC/FSE 2005*. New York NY: ACM Press, pp. 1–10, invited paper. [Online]. Available: http://scg.unibe.ch/archive/papers/Nier05cStoryOfMoose.pdf

[3] S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz, "A meta-model for language-independent refactoring," in *Proceedings of ISPSE 2000*. IEEE Computer Society Press, pp. 157–167. [Online]. Available: http://scg.unibe.ch/archive/papers/Tich00bRefactoringMetamodel.pdf

[4] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison Wesley, 2005.

[5] L. G. DeMichiel, "Enterprise JavaBeans specification, version 2.1," Sun Microsystems, Nov. 2003.

[6] M. K. Linda DeMichiel, "JSR 220: Enterprise JavaBeans specification, version 3.0," Sun Microsystems, May 2006.

[7] C. Marinescu, R. Marinescu, P. Mihancea, D. Ratiu, and R. Wettel, "iPlasma: An integrated platform for quality assessment of object-oriented design," in *Proceedings of ICSM 2005*, pp. 77–80, tool demo.

[8] P. Bunge, T. Gîrba, L. Renggli, J. Ressia, and D. Röthlisberger, "Scripting browsers with Glamour," European Smalltalk User Group 2009 Technology Innovation Awards, Aug. 2009, glamour was awarded the 3rd prize. [Online]. Available: http://scg.unibe.ch/archive/reports/Bung09bGlamour.pdf

[9] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice*. Springer-Verlag, 2006. [Online]. Available: http://www.springer.com/alert/urltracking.do?id=5907042