Enabling the evolution of J2EE Applications through reverse engineering and quality assurance

Fabrizio Perin

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

***Abstract*—Enterprise Applications are complex software systems that manipulate much persistent data and interact with the user through a vast and complex user interface. In particular applications written for the Java 2 Platform, Enterprise Edition (J2EE) are composed using various technologies such as Enterprise Java Beans (EJB) or Java Server Pages (JSP) that in turn rely on languages other than Java, such as XML or SQL. In this heterogeneous context applying existing reverse engineering and quality assurance techniques developed for object-oriented systems is not enough. Because those techniques have been created to measure quality or provide information about one aspect of J2EE applications, they cannot properly measure the quality of the entire system. We intend to devise techniques and metrics to measure quality in J2EE applications considering all their aspects and to aid their evolution. Using software visualization we also intend to inspect to structure of J2EE applications and all other aspects that can be investigate through this technique. In order to do that we also need to create a unified meta-model including all elements composing a J2EE application.**

***Keywords*-Reverse engineering; Java Enterprise; Software Visualizations; Software Metrics.**

I. Introduction

Since Java 2 Platform Enterprise Edition (J2EE) was introduced in 1999 it has become one of the standard technologies in enterprise application development. J2EE applications are complex systems composed using various technologies that in turn rely on languages other than Java, such as XML or SQL. In this context where the information is spread across source code, application deployment descriptors, JSP and other locations, applying existing reverse engineering and quality assurance techniques developed for object-oriented systems is not enough. All of those techniques are only able to look at the system from one angle without a vision on the entire system. This project aims to conduct a systematic study in reverse engineering and quality assurance of J2EE applications. In particular, we target the following questions:

1) How do we model J2EE to support analysis of the different languages?
2) What defines internal quality in J2EE applications and how do we measure it?
3) How do we visualize the diversity of languages to support the understanding of J2EE applications?

In the following sections we discuss briefly these questions providing samples and proposing solutions.

II. J2EE Meta-Model

A prerequisite to analyzing a system is an explicit meta-model to represent it. J2EE applications are composed of different technologies such as EJB or Java Server Pages (JSP). Often J2EE applications interact with a database, which means that they incorporate two different paradigms: the object-oriented paradigm for development and the relational paradigm in order to provide data persistence. Consequently, a meta-model for object-oriented systems is not enough to properly represent a J2EE application. We base our work on FAMIX [1], a language independent meta-model for representing and analyzing object-oriented software. The new meta-model expresses and analyze several issues: dependencies between Beans, dependencies between JSP and Java source code, HTML anchors that point from one JSP to another, database structure, dependencies between Java source code and database structure. We added to FAMIX a meta-model for EJBs and a meta-model for relational databases proposed by Marinescu [2]. This new parts have been linked to the already present classes and method entities to represent the relation between the EA and EJBs and the database.

In Figure 1 a simplified FAMIX meta-model is shown where the new parts are highlighted in bold. The new hierarchy that represents EJBs is in the top-left part of Figure 1. A generic Java Bean is related to the class that implements it. The deployment descriptor contains the mapping between the Beans 2.1 and the classes in the source code as well as other information. Starting from EJB version 3.0 [3], the deployment descriptor has been replaced by Java annotations.

Another aspect that is important to consider is the relation between the application and the database. Marinescu tackled this problem by presenting a meta-model to represent a relational database and how it can be relate and enriched with a meta-model for enterprise applications [4], [5],

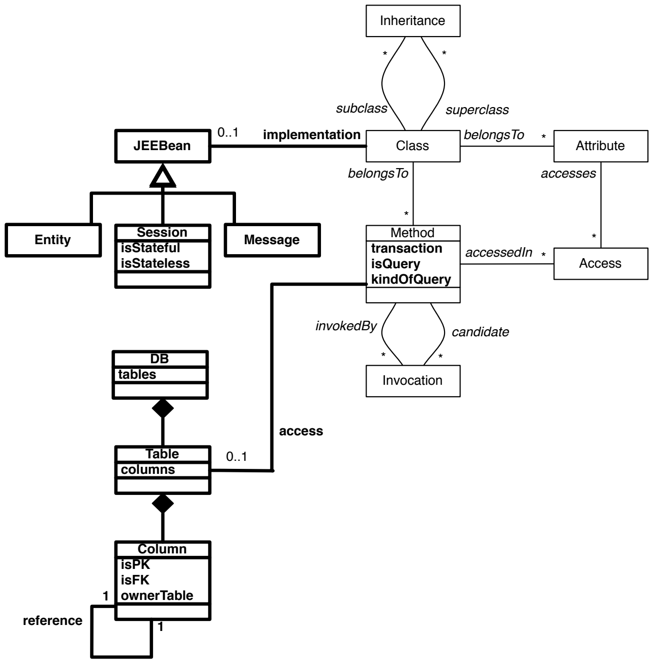

Figure 1. Enriched FAMIX Meta-Model

[2]. Also Keller investigated the relations between object-oriented software and relational databases providing some patterns describing this family of problem [6]. The new hierarchy representing a relational database is shown in the bottom-left part of Figure 1. The entity table is linked to the method that access it. Thanks to these modification we are able to connect JEAs elements with the database accessed by the application. For instance is possible to identify which database tables are accessed by which EJBs, that are the entry points for application's services. It is also possibles get useful information to identify all the components that related each others in order to drive modifications of a service.

III. J2EE METRICS AND QUALITY

In order to assess the quality of J2EE applications we intend to proceed in two way: defining metrics to measure different aspects of this kind of applications and identifying pattern for development.

J2EE applications are naturally subdivided into layers [7]. For example it is possible to calculate the number of classes that are part of a layer. The service layer should contain just classes used like entry points for services. If too many classes are in the service layer, this could be symptom of a bad design because it means that may the functionalities or services are split in the wrong way.

Other metrics like the average NOM per class and the average LOC per class [8] can be used to identify Session Beans that implement logic that should be defined in classes belonging to the business layer.

Another task that can be done using metrics is measuring the coupling between layers and to identify the layer's entry points. Other examples are: the number of Java source code lines in the JSP, the number of direct SQL queries both in JSP and in Java source code, the number of Beans. Some metrics that measure different aspects of the industrial case study that we are are analyzing are shown as an example in Table I.

Metric name	Old version	New version
NOC	1938	1527
NOCISB	45	38
A_NOM per bean	14.29	13.15
A_NOTM	0.928	0.849

Table I
SOME SIMPLE METRICS.

In Table I NOC is Number Of Classes, NOCISB is Number Of Classes Implementing a Session Bean, A_NOM per bean is the Average Number Of Method per bean and A_NOTM is the average number of methods that is involved in an application transaction.

The meta-model have to be populated with the information contained from all possible sources like the deployment descriptor. In this way it is possible to identify all methods that are involved in an application transaction, used to define atomic unit of work that can be rollbacked in case of application failure. A metric like A_NOTM can show wether or not the application's transaction scope properly cover the operations performed by the application or not.

We intend to catalogue some classic metrics that can be applied to different layers. Also we want create new metrics to measure the coupling between application's parts.

The definition of metrics to measure important aspects of J2EE applications is a problem related with the data that we can recover from them. We can collect low level information from Java classes, JS Pages or the data base like directory structure, data base schema or relation among JSP and Java classes or JSP and database. Those information are useful to understand the structure of the application. To define which data is needed for a concept in a higher level of abstraction we should define first which concept we intend to inspect. For instance, application transactions is a feature of J2EE application that implies the necessity to collect information from the deployment descriptor or from Java annotations as we said before.

There are several aspects to inspect, one of this could be the security. How to define the security level in a J2EE application could not easy, but it is possible to start for instance checking if in the application the Security Java API is used and in which way. Or may checking if there are some kind of rules to access services in the front-end. Reasoning from those elements may will be possible to define a security level rank.

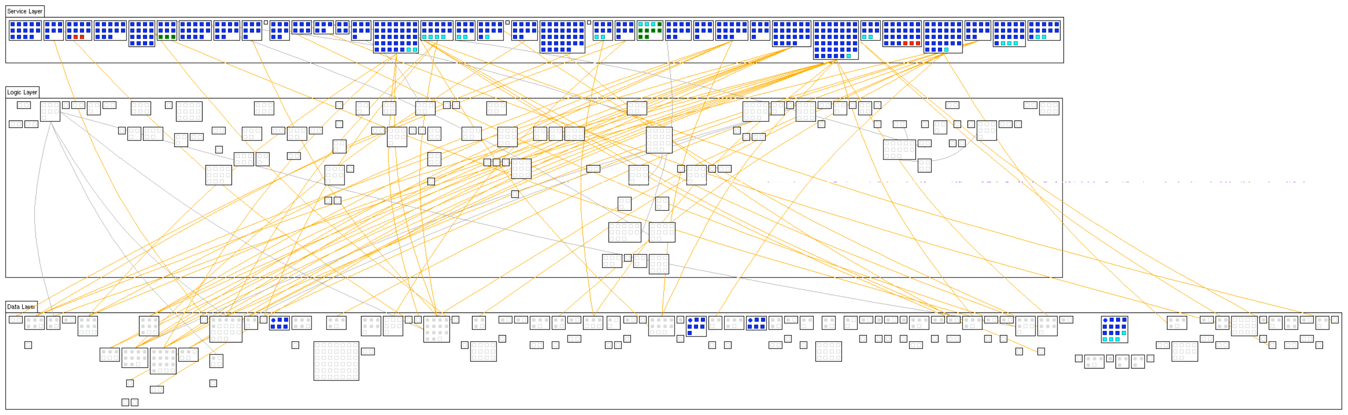

Figure 2. System split in layers.

There is a large body of development patterns and heuristics gathered by the engineering community from experience. That has been the starting point to study and to detect the existing patterns for enterprise applications [7] in general and patterns for J2EE [9] in particular. The description of design patterns provides information about the structure, the participant's roles, the interaction between participants and, above all, the intent for which they should be used. To identify a pattern means to identify a structure that is extendable. This is a synonymous of a code easier to maintain. Information related to the presence of a pattern is useful to understand not only the code, but also to realize the concepts behind its design. This has a significant implication for further improvement or adaptive changes and to identify classes that provide a particular service. For example, identifying a Table Data Gateway pattern it is easy to find all classes that invoke methods of the class used like gateway. In case of a modification of the database became easier to identify all classes that potentially should be modified.

Patterns for enterprise applications are naturally divided into layers [9]. To identify a pattern also means to identify which classes are belong to a particular layer. This could help to better understand the structure of the application.

IV. J2EE Visualizations

Visualizations are useful reverse engineering tools that can be used a first step in the process of designing automated detections. Visualizations are needed to create a unique overview on all parts composing a J2EE application. For example we can show components split into layers and the relationships between them, as well as components that access the database. With such a view we can possibly detect the dependencies that violate the layered architecture. In Figure 2 we show a visualization that gives an overview of the system split into layers [7]. The three layers shown are: Service, Logic and Data layer. Each layer consists of classes and their methods. Classes that contain colored methods are Session beans that should be part of the Service layer. In this case is immediately visible that four Session Beans are in the wrong layer. This happens because those beans access the database when they should not. In Figure 2 colored methods have a transaction attribute defined in the deployment descriptor. Thanks to this external information we can show all classes and their methods that are involved in a transaction. This kind of visualizations can also help to identify which classes are involved in the implementation of a service aiding the maintenance of the system.

This visualization could also to take JSP into account and their relation with the database. More specific visualizations are needed to inspect all parts of J2EE applications. The location in the file system of the various files is an important piece of information as it reveals the view of developers on the system. That is why we intend to construct a visualization that relates the source code to the file system to answer questions like: Which JSP files are grouped together? Are configuration files placed together with the source code they configure? One possibility is to use a visualization similar to the Distribution Map [10] that shows how different properties are spread over the software system. A Distribution Map shows partitions of the system (e.g. classes grouped in packages) and maps on the color the properties. After the creation of useful metrics we intend to create visualizations based on Polymetric Views [11]. The Polymetric View is a generic graph visualization that shows nodes as rectangles, and maps up to five metrics mapped on the dimensions (i.e. width and height), color and position of the rectangles (i.e. x and y). Considering the heterogeneous nature of J2EE applications, Polimetric Views could be a powerful instruments in the quality assessment process. Polymetric Views have been already used in several areas: showing the hierarchies of the system with System Complexity View [12], showing class internals with Class Blueprint [13] and others [14], [15]. All these applications have to be taken into account and also we intend to create other visualizations based on Polymetric Views specific for J2EE applications.

V. Significance of the research

We want to refine FAMIX by adding all parts and relations that are necessary to model a JEA. Having a consistent meta-model is possible to define on it a quality model based on metrics and pattern detection. We will proceed creating a catalogue of classic metrics that can be used in the context of J2EE and creating new metrics to measure different aspects of JEAs.

It is important take into account the relations among all parts of a J2EE application as well as its single components. Pattern identification will be a key point to analyze the structure of JEAs and to assess its quality. We also need many different visualizations builded using existing techniques and exploring new way to visualize J2EE applications structure and all helpful aspect that can aid the system maintenance. The final objective is to create a suite of techniques, metrics and visualizations to automatically, or semi-automatically, assess the quality of J2EE applications and enterprise applications in general and aid their evolution.

In order to validate the work we intend to be aided by the company that provide us case studies. We will perform human subject testing with expert of the specific case study and software engineer in general. Presenting them our results periodically to check if their are useful. We can collect their comments on visualizations or on metrics due to modify what we have done and to focus our work on new aspects of enterprise applications.

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 would also want to thank Tudor Gîrba, Oscar Nierstrasz, Jorge Ressia and Toon Verwaest for their comments on this paper and their support on this project.

References

[1] S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz, “A meta-model for language-independent refactoring,” in *Proceedings of International Symposium on Principles of Software Evolution (ISPSE ’00)*. IEEE Computer Society Press, 2000, pp. 157–167. [Online]. Available: http://scg.unibe.ch/archive/papers/Tich00bRefactoringMetamodel.pdf

[2] C. Marinescu and I. Jurca, “A meta-model for enterprise applications,” in *SYNASC ’06: Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 187–194.

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

[4] M. C., “Identification of Relational Discrepancies between Database Schemas and Source-Code in Enterprise Applications,” in *Symbolic and Numeric Algorithms for Scientific Computing, 2007. SYNASC. International Symposium on*, Sep. 2007, pp. 93–100.

[5] C. Marinescu, “Identification of design roles for the assessment of design quality in enterprise applications,” in *Proceedings of International Conference on Program Comprehension (ICPC 2006)*. Los Alamitos CA: IEEE Computer Society Press, 2006, pp. 169–180.

[6] W. Keller, “Mapping objects to tables - a pattern language,” in *Proc. Of European Conference on Pattern Languages of Programming Conference EuroPLOP ’97*, 1997.

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

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

[9] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*. Pearson Education, 2001.

[10] S. Ducasse, T. Gîrba, and A. Kuhn, “Distribution map,” in *Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM ’06)*. Los Alamitos CA: IEEE Computer Society, 2006, pp. 203–212. [Online]. Available: http://scg.unibe.ch/archive/papers/Duca06cDistributionMap.pdf

[11] M. Lanza and S. Ducasse, “Polymetric views—a lightweight visual approach to reverse engineering,” *Transactions on Software Engineering (TSE)*, vol. 29, no. 9, pp. 782–795, Sep. 2003. [Online]. Available: http://scg.unibe.ch/archive/papers/Lanz03dTSEPolymetric.pdf

[12] M. Lanza, “Object-oriented reverse engineering — coarse-grained, fine-grained, and evolutionary software visualization,” Ph.D. dissertation, University of Bern, May 2003. [Online]. Available: http://scg.unibe.ch/archive/phd/lanza-phd.pdf

[13] S. Ducasse and M. Lanza, “The class blueprint: Visually supporting the understanding of classes,” *Transactions on Software Engineering (TSE)*, vol. 31, no. 1, pp. 75–90, Jan. 2005. [Online]. Available: http://scg.unibe.ch/archive/papers/Duca05bTSEClassBlueprint.pdf

[14] M. Lanza and S. Ducasse, “Understanding software evolution using a combination of software visualization and software metrics,” in *Proceedings of Langages et Modèles à Objets (LMO’02)*. Paris: Lavoisier, 2002, pp. 135–149. [Online]. Available: http://scg.unibe.ch/archive/papers/Lanz02aEvolutionMatrix.pdf

[15] T. Gîrba, M. Lanza, and S. Ducasse, “Characterizing the evolution of class hierarchies,” in *Proceedings of 9th European Conference on Software Maintenance and Reengineering (CSMR’05)*. Los Alamitos CA: IEEE Computer Society, 2005, pp. 2–11. [Online]. Available: http://scg.unibe.ch/archive/papers/Girb05aHierarchiesEvolution.pdf