Evaluating Code Duplication to Identify Rich Business Objects from Data Transfer Objects

Fabrizio Perin and Tudor Gîrba

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

***Abstract*—Java Enterprise Applications (JEAs) are complex software systems written using multiple technologies. Moreover they are usually distributed systems and use a database to deal with persistence. A particular problem that appears in the design of these systems is the lack of a rich business model. In this paper we propose a technique to support the recovery of such rich business objects starting from anemic Data Transfer Objects (DTOs). Exposing the code duplications in the application's elements using the DTOs we suggest which business logic can be moved into the DTOs from the other classes.**

Keywords: Reverse engineering, Java, Enterprise Application, Software Visualization.

I. Introduction

Enterprise Applications are complex, large and usually distributed systems that handle large amounts of data. Even though these systems are typically implemented using a base language, such as Java, the complexity of these systems requires new kinds of concepts and technologies. The ecosystem around Java Enterprise Edition (JEE) offers several technologies to accommodate the technical requirements of such systems.

Two key design constraints are given by the distribution of the remote services and the persistency of data. In JEE, this is typically supported through SessionBeans and EntityBeans respectively [1] [2]. When two services can live on two different machines, they need to exchange data remotely. To make this dialog efficient, these services typically bundle the data in one carrier and transmit it in bulk. Such a carrier is called Data Transfer Object (DTO). Fowler describes such a DTO as *"an object that carries data between processes in order to reduce the number of method calls."* [3]

While these technologies do offer strong technical support, in many systems it can be noticed how even if the base language is called object-oriented, the design of the resulting system is very much procedural: the behavior typically lies in the SessionBeans, while the data lies in the EntityBeans and DTOs. These systems would benefit from a refactoring towards a stronger business model that unifies behavior and data and to which SessionBeans would delegate responsibilities.

In this paper, we propose a visualization that helps in the initial phase of identifying such a rich object model. Our strategy is to expose the DTOs and the way they are used by the different services. In particular, we are looking at two different relationships: the calls between services and DTOs and the code duplication in these classes.

II. DTOs Analysis and Evaluation

Code duplications between classes sharing the same DTOs can be a sign that certain business logic can possibly be factored out, either in the DTO, or in another class. In order to evaluate the effectiveness of this hypothesis we create a novel software visualization, called DTO Constellation.

Figure 1 shows the visualization as applied to an industrial content management system (CMS) to manage customer data. Gray squares represent DTOs while white squares are Java classes that invoke some methods of the DTOs. The light gray edges represent an invocation from one class to another while black edges represent code duplication. The graph is then laid out using a force based layout that reveals clusters of connected entities.

The visualization is drawn using the Mondrian visualization engine [4], while the code duplication is retrieved by SmallDude, a duplication detector included in the Moose analysis platform [5].

In our example, the DTOs are visually clustered in 3 groups each composed of several classes. Manual inspection of the code verifies that these elements actually have a common context and all together collaborate to manage logically related data.

Interestingly, the classes in the middle of Figure 1 are connecting two different clusters of elements. These classes are actually Session Beans that need to collect data from two different contexts exposing a relation between them that would not be evident otherwise.

The most interesting exposed elements are the code duplications expressed by the black edges. There are three kinds of code duplication present: (1) duplication between classes using the same DTO, (2) duplication between DTOs, and (3) duplications between classes of two different clusters.

The first kind of duplication suggests the presence of code that could be factored out from the classes suffering the duplication.

The second kind of duplication can highlight the presence of DTOs sharing the same data or sharing the same logic to manage different data. A further inspection of these elements

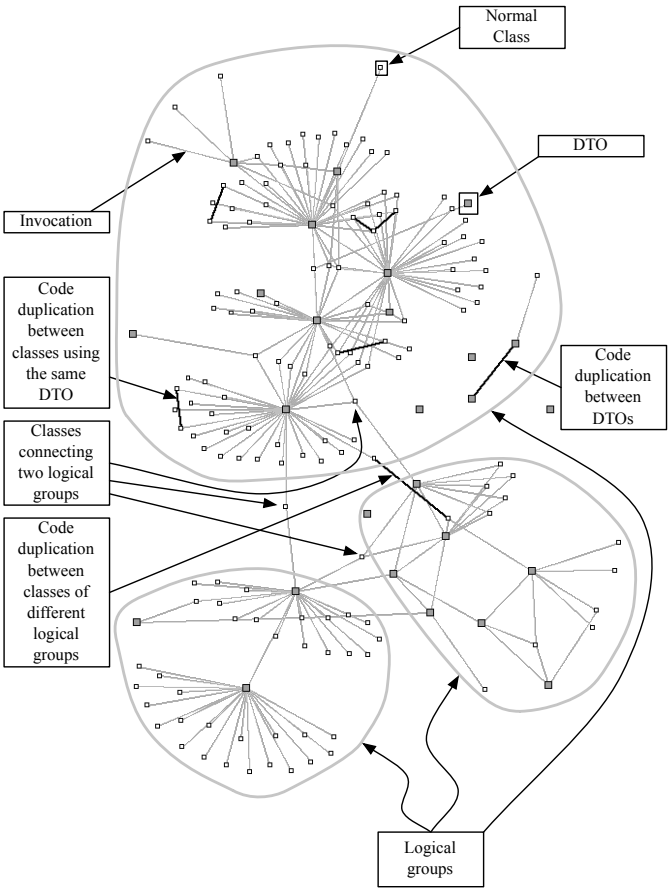

Fig. 1. Prototype of Visualizations to highlight the relations between DTOs and to inspect the code duplications

can drive the merging of two DTOs into one or the redistribution of the management of the data and the data itself.

By investigating non-evident relations among software elements and using this information to drive the refactoring, the third kind of code duplication can highlight the presence of the same business logic being used to manage different kind of data.

The duplications between classes using the DTOs relate to some shared business logic that could be factored out. In contrast, the duplication between two DTOs reveal shared data that can be factored out in a smarter DTO hierarchy.

III. Conclusions and future works

In this paper we propose to support the refactoring towards the creation of rich business in an Enterprise Application. We achieve this by analyzing the DTOs and their relations with other classes. To this end, we visualize the calls and the code duplications present in the classes related with the DTOs. We applied it to an industrial case study, and we were able to identify three logical sets of data objects, to identify three kind of duplication, and to identify SessionBeans that used data from two disjoint group of elements.

We plan to continue in this direction, and to refine the presented technique and visualization. We plan also to perform other kind of analyses on DTOs.

First, we intend to evaluate the relations between and EntityBeans and DTOs. By evaluating the relations among the DTOs, the Entity Beans and the database it is possible to see which are the data used by a DTO. We can match the database tables to the DTOs that use them, thus logically grouping these tables. We can also evaluate which data from the database tables are actually used, leading to an analysis of the organization of the data in the database.

Second, we plan use static data flow analysis [6] to trace a DTO in the source code to highlight how the actual data moves inside the application.

Third, exposing the relations among DTOs and other software element can help us to logically group application's elements that share the same data. Comparing this clustering with the existing static package organization or the run-time architecture can provide helpful insight when refactoring the application's structure.

In order to validate this work we plan to perform experiments using industrial partners.

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, Jorge Ressia and Erwann Wernli for their comments on drafts of this paper.

References

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

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

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

[4] M. Meyer, T. Gîrba, and M. Lungu, "Mondrian: An agile visualization framework," in *ACM Symposium on Software Visualization (SoftVis'06)*. New York, NY, USA: ACM Press, 2006, pp. 135–144. [Online]. Available: http://scg.unibe.ch/archive/papers/Meye06aMondrian.pdf

[5] O. Nierstrasz, S. Ducasse, and T. Gîrba, "The story of Moose: an agile reengineering environment," in *Proceedings of the European Software Engineering Conference (ESEC/FSE'05)*. New York NY: ACM Press, 2005, pp. 1–10, invited paper. [Online]. Available: http://scg.unibe.ch/archive/papers/Nier05cStoryOfMoose.pdf

[6] M. Hind, "Pointer analysis: Haven't we solved this problem yet?" in *2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'01)*. New York, NY, USA: ACM, 2001, pp. 54–61.