DATES: Design Analysis Tool for Enterprise Systems

Cristina Marinescu
LOOSE Research Group
“Politehnica” University of Timişoara, Romania
cristina.marinescu@cs.upt.ro

Abstract

*In the current demonstration we present a new tool which increases the level of understanding and the accuracy of design quality assessment within enterprise systems. This is performed by providing its users with information specific to this type of systems (*e.g.*, accessed tables from a class). In order to validate its usefulness, we perform some experiments on a suite of enterprise systems whose results are briefly presented in the last part of the demo.*

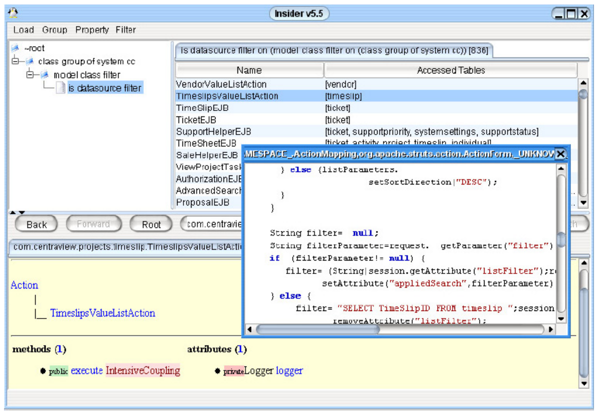
1. Introduction

Understanding and assessing the design quality of enterprise software systems using tools that consider an enterprise system as a *regular* [1] object-oriented one are in danger of getting *incomplete* and *inaccurate* results. The two limitations reside in the fact that the aforementioned tools:

- rely on a structural view of the source code – this led to an impossibility of finding, for example, object-oriented design entities which access relational design entities (*e.g.*, which classes are affected when we change the structure of a particular table?).
- rely exclusively on principles, heuristics and best practices of object-oriented design, being impossible: (1) to find the design entities that break specific enterprise systems design rules and patterns like the ones defined in [4, 9], (2) to determine which object-oriented design entities are/are not affected by conflicting bad smells (*e.g.*, a Data Class bad smell [3] within an enterprise system may not be a design bad smell).

In the current demonstration we present a new tool called DATES whose role is to increase the level of understanding and the accuracy of design quality assessment within enterprise systems.

1 an object-oriented system which does not involve persistency (currently, provided by a relational database).



is datasource filter on (model class filter on (class group of system cc)) [836]

Name	Accessed Tables
VendorValueListAction	[vendor]
TimeslipsValueListAction	[timeslip]
TimeSlipEJB	[ticket]
TicketEJB	[ticket]
SupportHelperEJB	[ticket, supportpriority, systemsettings, supportstatus]
TimeSheetEJB	[ticket, activity, project, timeslip, individual]
SaleHelperEJB	
ViewProjectTask	
AuthorizationEJB	
AdvancedSearch	
ProposalEJB	

MESPACE_.ActionMapping,org.apache.struts.action.ActionForm,_UNKNOV

```
    } else {listParameters.
                    setSortDirection("DESC");
    }
  }

  String filter=  null;
  String filterParameter=request.  getParameter("filter")
  if  (filterParameter!= null) {
     filter= (String)session.getAttribute("listFilter");r
         setAttribute("appliedSearch",filterParameter)
  } else {
       filter= "SELECT TimeSlipID FROM timeslip ";session
         removeAttribute("listFilter");
```

Figure 1. DATES - a snapshot.

2. DATES – an overview

In this section we show that DATES removes the mentioned limitations existing tools for reengineering regular object-oriented systems (*e.g.*, MOOSE[2], iPlasma [7]) are confronted with when applied on enterprise systems. Next, for each described limitation we dedicate a section.

Finding accesses from object-oriented entities to relational entities. Due to the fact that DATES uses an enhanced model of the analyzed source code (presented in [6]) which includes design entities from the object-oriented part, design entities from the involved relational database as well as the interactions among them, it is possible to find out:

- the unused tables from the involved relational database.
- the tables accessed by classes and methods from the data source layer (the upper right part of Figure 1).

Finding entities that break specific enterprise systems design rules. As stated in [4], one steady rule an enterprise system should follow is that its domain and data source layers should never depend on the presentation layer. DATES allows us to detect entities which have a double identity (belong to more than one layer), according to a lightweight approach of layers'identification presented in [5].

In [4] are also presented specific design patterns regarding the communication between the object-oriented entities and the relational entities (*e.g.*, Table Data Gateway, Active Record). DATES allows us to identify classes whose structure respect/do not respect the aforementioned patterns according to the approach presented also in [5].

Finding entities that are/are not affected by conflicting bad smells. One of the well-known design flaws is the *Data Class* [3] design flaw. But within an enterprise system, Data Classes used for carrying information between clients (the domain layer) and server (the persistency layer) in order to reduce the number of method calls are not considered harmful (these classes are called *Data Transfer Objects* [4] (DTO)). DATES does not report DTO classes as regular Data Classes that should be refactored.

Having some Data Transfer Objects in an enterprise system probably involve the existence of some methods affected by the *Feature Envy* [3] design flaw. DATES, according to the approach from [5], allows us to determine which methods apparently affected by the Feature Envy design flaw should be refactored and which not (the ones being *envy* of data belonging to DTO classes).

3. DATES at work

DATES can be used for assessing enterprise systems written in Java where the persistency is provided by SQL relational databases, in particular MySQL and Microsoft Access. We integrate DATES within the iPlasma [7] front-end called *insider* in order to be able to use existing quality assessment techniques regarding the object-oriented part of an enterprise system (*e.g.*, metrics, detection strategies [8]).

When DATES is loaded into the insider front-end it enhances the model of the analyzed system with information regarding the relational part of the system, as well as with information regarding the interactions between the two involved paradigms. It also overrides the existing implementations of static analyses (for regular object-oriented systems) for finding entities affected by Data Class and Feature Envy design flaws with the ones suitable for enterprise systems presented in [5].

We analyzed with DATES several enterprise software systems, among which an open-source enterprise system having 11.2 MB, 175424 LOC, 1527 classes and 217 tables. The model of this enterprise system is extracted from the source code and database schema, using an Intel Core 2 Duo processor, in less than 2 minutes. The obtained results reveal, on the three fronts, the following:

- among the accesses from object-oriented entities to relational entities there are several false negatives caused by the existence of some accesses to tables which do not exist in the involved database.
- the systems contain classes which belong into more than one layer.
- an important number of classes initially affected by the Data Class design flaw are, in fact, Data Transfer Objects. DATES, by taking into account the difference between Data Classes and Data Transfer Objects, reports up to 81% of the methods affected by the Feature Envy design flaw as being non-harmful (*i.e.*, being *envy* of data belonging to DTO classes).

4. Related Work

There are already tools on the market that amongst other analyses are capable of analyzing the relationships between object-oriented and database design entities (*e.g.*, the CAST AI Platform [1]). In this context, we want to emphasize that DATES allows us to find valuable information related to enterprise systems which are not provided by other similar tools and, being a research infrastructure, its users are able to implement using a plugin mechanism further analyses that allows a better comprehension of enterprise software systems.

References

[1] *CAST AI Platform.* http://www.castsoftware.com, 2007.

[2] S. Ducasse, M. Lanza, and S. Tichelaar. Moose: an extensible language-independent environment for reengineering object-oriented systems. In *Proc. International Symposium on Constructing Software Engineering Tools*, 2000.

[3] M. Fowler. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

[4] M. Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley, 2003.

[5] C. Marinescu. Identification of design roles for the assessment of design quality in enterprise applications. In *Proc. IEEE International Conference on Program Comprehension*, 2006.

[6] C. Marinescu and I. Jurca. A meta-model for enterprise applications. In *Proc. International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC).* IEEE Computer Society Press, 2006.

[7] C. Marinescu, R. Marinescu, P.F. Mihancea, D. Raţiu, and R. Wettel. iPlasma: An integrated platform for quality assessment of object-oriented design. In *Proc. IEEE International Conference on Software Maintenance (Industrial and Tool Volume)*, 2005.

[8] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Proc. IEEE International Conference on Software Maintenance*, 2004.

[9] C. Nock. *Data Access Patterns: Database Interactions in Object-Oriented Applications.* Addison-Wesley, 2003.