UNIFYING SOFTWARE AND DATA REVERSE ENGINEERING
A Pattern based Approach

Francesca Arcelli, Gianluigi Viscusi and Marco Zanoni
Dipartimento di Informatica Sistemistica e Comunicazione, Università degli Studi di Milano Bicocca
Viale Sarca 336, Milano, Italy

Keywords: Data reverse engineering, Design pattern detection, Persistence frameworks.

Abstract: At the state of the art, objects oriented applications use data structured in relational databases by exploiting some patterns, like the Domain Model and Data Mapper. These approaches aim to represent data in the OO way, using objects for representing data entities. Furthermore, we point out that the identification of these patterns can show the link between the object model and the conceptual entities, exploiting their associations to the physical data objects. The aim of this paper is to present a unified perspective for the definition of an integrated approach for software and data reverse engineering. The discussion is carried out by means of a sample application and a comparison with results from current tools.

1 INTRODUCTION

The present paper discusses a first step towards the definition of an approach to reverse engineering, providing a unified perspectives on software and data. To this end, we first discuss an experience and a methodology for a repository based approach to data reverse engineering (DRE) (Davis and Aiken, 2000), and the main points of convergence with a design pattern based approach for reverse engineering (Serge Demeyer and Nierstrasz, 2008). The discussion of both the perspectives supports the motivations for the unified approach described in the paper.

The paper is structured as follows. Section 2 discusses related works, by focusing in particular on convergence between data and software reverse engineering. Section 3 describes design pattern detection for DRE. Section 4 discusses the proposed unified approach, by introducing an example of a real application. Concluding remarks and future work are discussed in Section 6.

2 RELATED WORK

In enterprise systems the separation between application and data logics is mandatory and effective only if both of them reach the same attention.

Nevertheless databases are often considered black boxes where applications have to act in a blind and costly way.

As noted in (J.L. Hainaut and Englebert, 2000) the understanding of data structures and of programs that manipulate them are strictly tied in providing the full functional specifications of an information system. Nevertheless, methodologies and approaches providing a unified perspective have been poorly investigated in the literature; while, for example, considering only DRE, at the state of the art strategies, methodologies, and tools for DRE have been proposed and discussed (J.L. Hainaut and Englebert, 2000; Mian and Hussain, 2008).

In this paper we aim to point out that DRE provides not only the conceptual schemas allowing better data governance, but also a domain model that can be accessible at the application layer. The analysis of such a convergence has been investigated at the state of the art for example i) for detecting relational discrepancies between database schemas and source-code in enterprise applications, and ii) correlating the information extracted from the database schema with the usage of the database elements within the source code (Marinescu, 2007).

Furthermore, objects oriented applications use data structured in relational databases by exploiting some known patterns, like the *Domain Model* and *Data Mapper* patterns (Fowler, 2002).

The identification of patterns involved in data

Arcelli F., Viscusi G. and Zanoni M. (2010).
UNIFYING SOFTWARE AND DATA REVERSE ENGINEERING - A Pattern based Approach.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 208-213
DOI: 10.5220/0003010202080213

management represents a way of realizing a particular and important type of application exploration. Beside, the conceptual schemas of the databases provide a domain model nearest to the applications model domain. Unfortunately conceptual schemas represent a rare resource within organizations, and in particular in large organizations with different and large information sources, including not only relational databases. Motivations for such a rarity depends on different factors: age difference between the deployment of various legacy systems, lack of design skills and capabilities in developers, time-to-market issues leading to a poor attention to the design phase, major focus on applications and relevance of what we can call *transaction script* (Fowler, 2002) attitude, etc.

Exploiting patterns like *Domain Model* and *Data Mapper* it is possible to represent data in the object oriented way, using objects for representing data entities. These patterns are the glue between software and data in object oriented systems. We argue that the identification of these patterns shows the link between the object model and the conceptual entities, walking through their associations to the physical data objects (i.e. relational tables, XML entities). In this way it is possible to infer the knowledge gained about the data and transfer it to the application, or vice-versa.

3 DESIGN PATTERN DETECTION FOR DRE

The development of an enterprise system is a very complex task; for this reason software engineers decide to use architectural patterns for the design phase. In the literature several architectural patterns have been proposed (Fowler, 2002) and in particular there are some architectural patterns designed for the interaction between the application and persistent data.

In this kind of applications the most common view is the Data-centered view (Avgeriou and Zdun, 2005) that sees the system as a persistent, shared data store that is accessed and modified by a certain number of elements. In this view we can find at least three architectural patterns: *Shared Repository*, *Active Repository* and *Blackboard*.

The identification of this kind of patterns allows us to identify the link between the application and persistent data and through it we can increase the knowledge on both the application and the persistent data.

A well-known and widely used Enterprise Design Pattern has been proposed as a specialization of the Shared Repository architectural pattern, and it is very interesting in the Reverse Engineering phase: the Data Access Object Pattern (DAO) (Alur et al., 2001). It is defined as a way to abstract and encapsulate all access to the data source. It manages the connection with the data source to obtain and store data.

If engineers know about the presence and the location of that pattern instance, he can directly know the connected domain entity (business object). The identification of the pattern instances is not very complex because DAO patterns are often organized using a *Factory Method* or *Abstract Factory* design pattern, so they belong to well-organized structures and looking at factories it is possible to know all of them.

In the context of data-oriented patterns other patterns have been proposed, for example in (Fowler, 2002), the author addresses many type of problem and defines a set of patterns which are currently used in many widely adopted persistence layer frameworks. The identification of those framework instances can be a good hint about the identification and management of persistent entities; in addition a deeper analysis can also reveal which type of technology has been used in order to implement the persistence layer.

4 THE UNIFIED APPROACH

The detection of data-related patterns or frameworks (see Section 3) in the application can lead to the identification of the domain model entities, looking at the upper application layer, and to the connection of each of those entities to the underlying relational table/s (in the case of a relational database physical data layer). Combining the gained knowledge carried out with data reverse engineering techniques (seee Section 2), that abstracts a relational database to a conceptual schema, it is possible to associate conceptual data entities to the object model entities, putting together the knowledge about the two separated layers. A consequence of these merging is for example the transfer of the abstraction hierarchy defined on schemas to the domain model, permitting an higher-level view of the model itself. More generally this approach will bring us towards an integrated view of application entities and data entities, and their interdependencies. An example of the approach is represented in Figure 1, that shows how the entity *e5* is recognized to be associated to the domain object *do5*, because they are both associated to table *t4*.

In this Section we discuss the steps of the proposed unified approach. Considering the information system of an hypothetical organization, the starting input are (i) the available classes (software perspective), and (ii) the legacy database(s) (data perspective). Data and Software Reverse Engineering look for similar concepts in data and software schemas, in

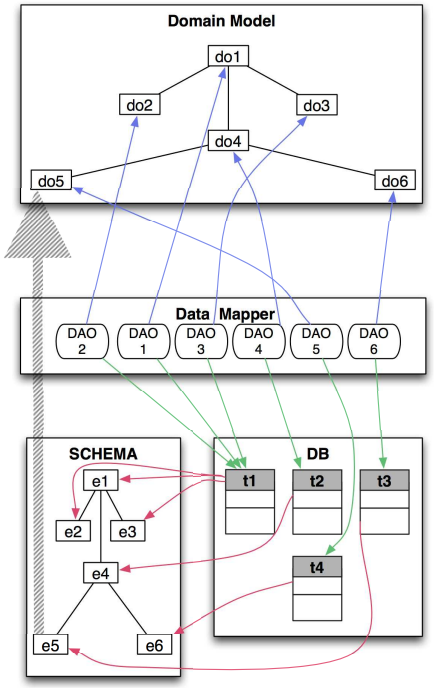

Figure 1: An example of integrated analysis.

order to evaluate their completeness and modify the overall information system.

In order to provide a unified representation at conceptual level, the proposed approach includes the following steps:

1. *Code analysis* in order to retrieve the software structure and patterns.

1.1. *Extract structure* in order to retrieve software classes, methods, generalizations, references.

1.2. *Extract patterns* that represent potential use of data entities,(e.g DAO, persistence layer, ecc.).

2. *Extract data entities and relationships* from logical/physical schemas analysis, producing a first skeleton of conceptual schema.

3. *Compare* classes representing potential data entities with data entities, producing a unified schema (see Figure 1).

3.1. *Fill the gap* between data and application layer adding additional knowledge gained on the data to the application and viceversa.

Once the conceptual schemas have been reengineered it is possible to build the abstract schemas by applying methods such as, for example, the ones described in (Batini et al., 2005).

5 TOOL EXPERIMENTATION

In order to better explain our approach we experiment three tools on a sample application, comparing the results of these tools with the results of our methodology. The sample application uses DAO to access to a database. The target conceptual schema of the database is shown in Figure 2.

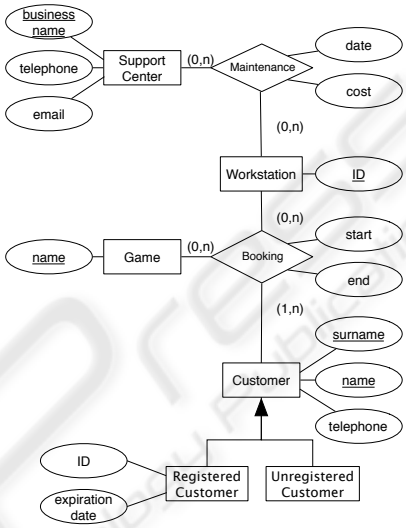

Figure 2: The conceptual schema of the example.

The logical schema (see Figure 3) we use for the example, derived from the conceptual schema, has two important characteristics:

- the generalization was implemented using three tables: this is a reasonable choice, because the `Unregistered Customer` entity has no attributes, while `Registered Customer` has two attributes, and the `Customer` entity is involved in the `Booking` relationship;
- the foreign key between `workstation` and `maintainance` and the one between `supportcenter` and `maintainance` are omitted: this choice makes the example nearer to a real scenario that can miss some foreign key.

In order to perform the experiment we used a mysql RDBMS version 5.1 on a windows xp system. The tools we experimented are:

- **CA ERwin Data Modeler.** Version 7.3
- **IBM Rational Data Architect.** Version 7.5
- **Embarcadero ER/Studio.** Version 8.0

Next we describe the results of the three tools, the results of the application of our methodology showing the steps of our approach on this example and a brief comparison of the results.

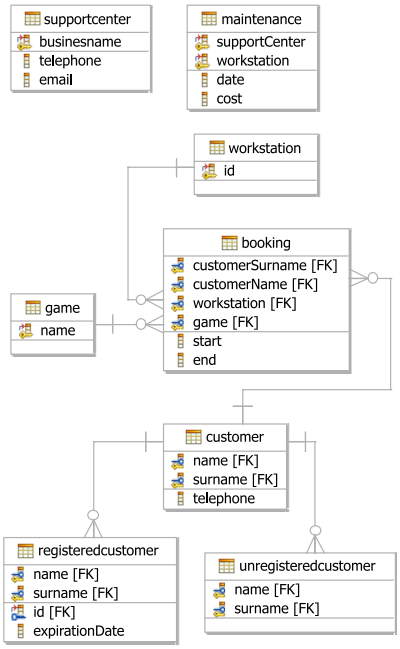

Figure 3: The logical schema of the example.

5.1 Tools Results

The performance of the three tools are similar, they are able to reconstruct only the logical schema and they perform foreign key elicitation under very strong assumptions: primary key and foreign key must have same names and types; presence of indexes.

For this reason (as shown in Figure 5 (ERwin), 4 (Rational Data Architect) and 6 (ER/Studio)) the tools were not able to reconstruct hierarchy relationships and they also were not able to reconstruct the relationships between `workstation` and `maintainance` and the one between `supportcenter` and `maintainance`. The bold relationships in Figure 5 and 4 are the ones that not exist in the real schema. In the following we discuss results for the considered tools.

Rational Data Architect. This tool found some non-existent relationship (see Figure 4): for example it found every combination of relationships between `customer`, `registered customer` and `unregistered customer` because all the three table have the column `name` and `surname`.

ERwin. This tool found, like Rational Data Architect, some non-existent relationship (see Figure 5) but they were only four. It found for example, like Rational Data Architect, the relationship between `workstation` and `registered customer` because `workstation` has the column `ID` as primary key and `registered customer` has the column `ID` as an attribute.

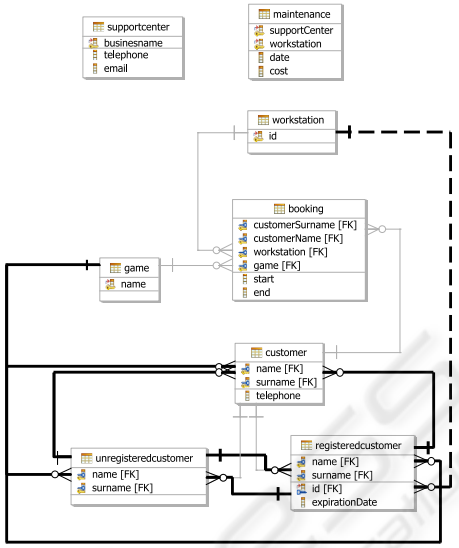

Figure 4: Rational Data Architect reverse engineering.

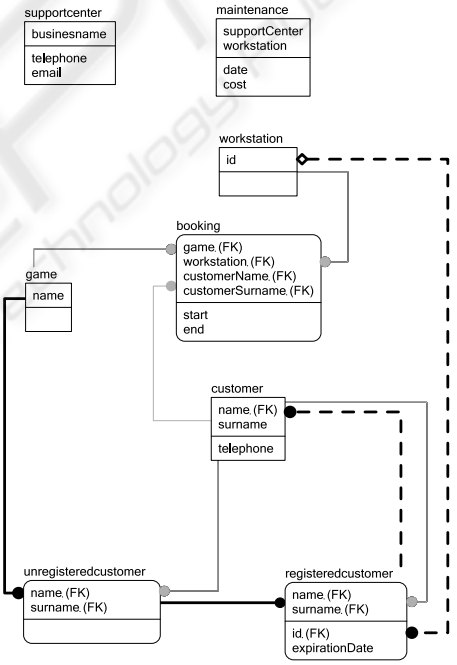

Figure 5: ERwin reverse engineering.

ER/Studio. This tool reconstructed only the declared relationships, even enabling foreing key inference by names and indexes (see Figure 6).

5.2 A Unified Methodology

In this Section we show our methodology applied to the same example used above.

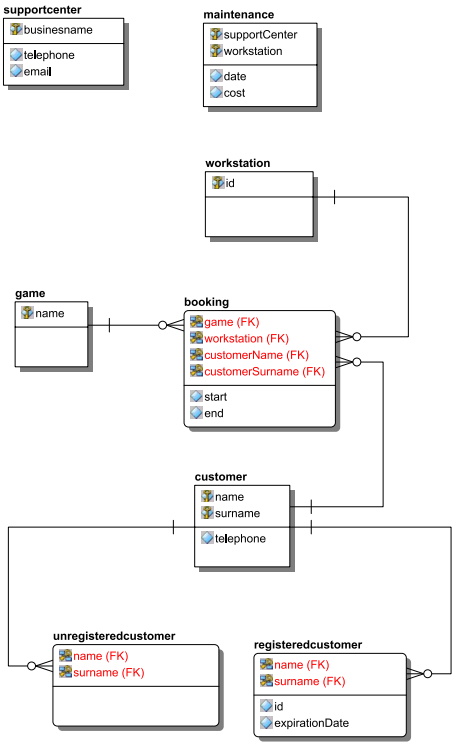

Figure 6: ER/Studio reverse engineering.

5.2.1 Code Analysis

We apply static analysis techniques to reconstruct the structure of the source code, following steps.

Extract Structure. This step collects source code elements, exploiting static analysis. Many tools and techniques are available to make this kind of analysis at this detail level, i.e. Doxygen [1], Rational Software Architect [2], Moose [3]. Through this analysis we are able to reconstruct the UML diagram of the system. In Figure 7 is visible the reconstructed domain entities' UML class diagram that contains: the relationships between `Workstation` and `Maintainance` and the one between `SupportCenter` and `Maintainance`; the generalization between `Unregistered Customer` and `Customer` and the one between `Registered Customer` and `Customer`.

Extract Pattern. This step uses the collected data and recognizes the use of the DAO pattern. In this example we perform a "manual" identification of the pattern because the aim of our example is not the identification of pattern but showing a metodology for data

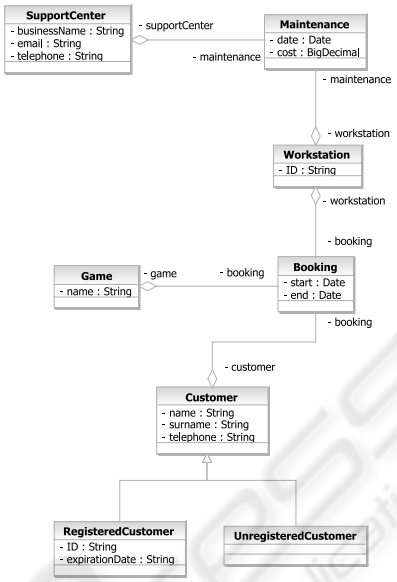

Figure 7: The domain entities' reconstructed UML class diagram.

and software reverse engineering. More in general using pattern matching techniques, like the ones we developed in our project Marple (Arcelli et al., 2008) for design pattern detection, it is possible to implement recognition rules for this pattern.

5.2.2 Extract Data Entities and Relationships

The database schema is analyzed in order to build a model keeping track of all the tables, with their attributes, keys, and foreign keys. In the database model there will be the two tables. For our purposes no inference is needed but it's sufficient to reconstruct only the existent relationship like in Figure 3.

5.2.3 Compare

This task focuses on the mapping of tables to entities. In this particular case we must find the relationships between DAO implementations and tables in the database. This phase depends on the way this mapping is done: for example we can use some persistence framework or directly the JDBC APIs (like in the example). However the identification of the used table is quite simple, through an automated or manual inspection of the DAO implementation.

Fill the Gap. Finally it is possible to see if the definition of the application model satisfies the data model constraints and viceversa. In this way we are able to detect the inconsistencies in the table definitions explained above:

[1] http://www.stack.nl/ dimitri/doxygen/

[2] http://www-01.ibm.com/software/awdtools/swarchitect/websphere/

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

- the relationships between `workstation` and `maintainance` and the one between `supportcenter` and `maintainance` are missing;
- the generalization between `unregisteredcustomer` and `customer` and the one between `registeredcustomer` and `customer` are missing.

6 CONCLUSIONS

In conclusion the example we described emphasizes the usefulness of our approach. This approach seems to be promising in the context of an integrated reverse engineering process of both data and software.

Furthermore, the detection of data related patterns could be done exploiting different tools for design pattern detection (DPD). We are working on the development of a tool for DPD called Marple (Metrics and Architecture Reconstruction PLugin for Eclipse)(Arcelli et al., 2008), where the detection of patterns is based on the recognition of micro structures that give useful hints of the presence of design patterns; the actual state of pattern detection technologies brings poorer information than the one extracted through the analysis of known persistence frameworks, due to the higher explicitly of the representation of the mappings, but this is the only fallback in absence of known libraries.

In future work we will provide an architecture for the proposed approach, implemented in Marple, as a first prototype integrating DAO and other data related patterns detection, and DRE techniques in order to exploit also the knowledge from available logical and conceptual schemas.

ACKNOWLEDGEMENTS

We acknowledge Christian Tosi for support in early application of the approach. We also acknowledge Prof. Carlo Batini for suggestions and discussion of the issues related to data reverse engineering.

REFERENCES

Alur, D., Crupi, J., and Malks, D. (2001). *Core J2EE Patterns: Best Practices and Design Strategies, 1/e*. Prentice Hall.

Arcelli, F., Tosi, C., Zanoni, M., and Maggioni, S. (2008). The marple project - a tool for design pattern detection and software architecture reconstruction. In *Proceedings of the WASDeTT Workshop, co-located event with ECOOP 2008 Conference*, Cyprus.

Avgeriou, P. and Zdun, U. (2005). Architectural patterns revisited - a pattern language. In *Proceedings of the 10th European Conference on Pattern Languages of Programs (EuroPLoP 2005)*, Irsee, Germany.

Batini, C., Garasi, M. F., and Grosso, R. (2005). Reuse of a repository of conceptual schemas in a large scale project. In *Advanced Topics in Database Research*. Idea Book.

Davis, K. H. and Aiken, P. H. (2000). Data reverse engineering: A historical survey. *Reverse Engineering, Working Conference on*, 0:70.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

J.L. Hainaut, J. Henrard, J. H. D. R. and Englebert, V. (2000). The nature of data reverse engineering. In *Data Reverse Engineering Workshop (DRE2000)*.

Marinescu, C. (2007). Discovering the objectual meaning of foreign key constraints in enterprise applications. In *Proceedings of WCRE 2007: 14th Working Conference on Reverse Engineering*, pages 100 –109.

Mian, N. A. and Hussain, T. (2008). Database reverse engineering tools. In *SEPADS'08: Proceedings of the 7th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*, pages 206–211. World Scientific and Engineering Academy and Society (WSEAS).

Serge Demeyer, S. D. and Nierstrasz, O. (2008). *Object-Oriented Reengineering Patterns*. Square Bracket Associates.