# Reengineering of Database Applications to EJB Based Architecture

Jianguo Lu

Department of Computer Science, University of Toronto
jglu@cs.toronto.edu

**Abstract** The advent and widespread use of Enterprise JavaBean (EJB) technology not only demands more reengineering support for legacy database applications, but also changes the reengineering practice. Initiated from our experience of reengineering database applications to EJB based architecture, this paper addresses two challenges in the mapping between database queries and EJBs. The first is to map a SQL to the equivalent EJB client code when the enterprise beans exist. The second is to generate enterprise beans from the set of legacy SQL expressions when the EJB architecture does not exist in the first place. We propose the EJB-SQL mediator to solve the first problem, and a view selection algorithm to solve the second one.

## 1 Introduction

There has been a great divide between the object world and the relational world. Both techniques are successful in the mainstream industrial practice, one for programming and the other for data management. In many cases these two worlds cohabitate peacefully, not interfering with each other. Unfortunately, many large applications, especially multi-tier e-commerce ones, need to use objects as the programming interface and use relations to manage the data. Current common practice in combining the two worlds is to imbed SQL expressions inside the classes. This close coupling of the object and relation runs against many software engineering principles, such as modularity, information encapsulation, usability, maintainability, etc.

One of the objectives of the Enterprise JavaBean (EJB) [34] technology is to address gracefully this object-relation interface. EJB can be seen as, among other things, the object view of the relational database system. With EJB, databases are transparent to application developers.

EJB based applications are seldom developed from scratch. In most cases they are reengineered from existing systems, especially the multi-tier applications that use relational databases as their back-ends. In particular, such legacy applications include large amount of SQL queries imbedded in various programming or scripting languages such as C++, or proprietary languages such as PL/SQL[18] and Net.Data [15]. While reengineering such systems to EJB-based architectures, there are three problems need to be addressed. First, how to map the relational schema to the object model in EJB. Second, how to translate the legacy queries to the EJB client code that access the enterprise beans. Third, when the enterprise beans do not exist, how to generate those beans and the behavior part of the EJB model from the set of queries of the legacy system.

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 361-376, 2002.

The first problem can be viewed as the traditional object-relational mapping problem that has been thoroughly studied for a long time [22] [3] [4]. Those studies have proposed a variety of methods and tools to support such a mapping, ranging from design patterns [5] to tools that generate the EJB skeletons from database schemas [17].

This paper focuses on the second and the third problems, which can be viewed as the other half of the object-relational mapping, i.e., the mapping between the relational queries and the behaviors of the objects. This part of object relational mapping has been barely touched in literature.

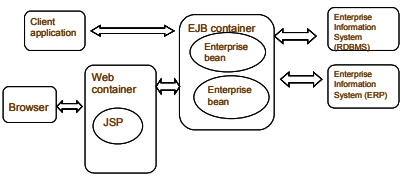


**Fig 1.** EJB architecture

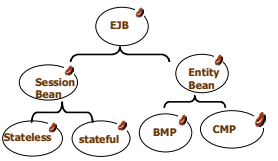


**Fig. 2.** Varieties of enterprise beans

By viewing EJBs as wrappers that represent the object views for the underlying database systems, the translation of a query to the EJB client code can be formulated as the query rewriting problem, a problem that has been widely studied [26][25]. Likewise, the EJB generation from legacy queries can be formalized as the view selection problem in databases [27].

In this paper, we propose an SQL-EJB mediator in order to generate the EJB client code. The mediator accepts a legacy query and uses descriptions of the query answering capabilities of available enterprise beans in order to translate that query into the equivalent EJB client code. The paper also proposes an enterprise bean generator that accepts a set of legacy queries as input, and produces a set of enterprise beans that can accommodate these queries.

Mapping SQLs into EJB architecture is an important problem for many reasons. Firstly, there are many legacy systems are being reengineered into EJB based applications. Secondly, the design of the EJB architecture itself should rely not only on the relational schemas of the underlying databases, but also on SQL expressions in the legacy system. In current practice, the methods attached to enterprise beans depend largely on developer experience. Providing developers with comprehensive information on the legacy queries that the EJB architecture will have to accommodate can facilitate the bean design process and make it less ad hoc. Thirdly, our research has implications not only in the reengineering of database applications to EJB architecture, but also for the developing of application that have object-relational mappings.

This paper is organized as follows. Section 2 introduces the background knowledge of EJB, Object-relational mapping and query rewriting. Section 3 introduces the overall architecture of the EJB reengineering problem. Section 4 describes the transformation of SQL expressions to client code of EJBs, supposing the EJB architecture already exists. Section 5 discusses the generation of EJB architecture from the legacy SQL expressions. In section 6 we explain the experiments in the IBM Websphere Commerce Suite reengineering project. Section 7 discusses related and future work.

# 2 Background

## 2.1 EJB Architecture

EJB is a distributed component framework that provides services for transactions, security and persistence in the distributed multi-tier environment [34]. It separates the business logic from the low-level details so that developers can concentrate on the business solution. With the growing popularity of the EJB framework, more and more legacy systems and old web-based systems are being transitioned into EJB architecture. Typically, in the legacy systems there are large amount of SQL queries. On the other hand, in the EJB based applications, those SQL queries are not directly used. Instead, by adopting the popular model-view-control design pattern, queries are wrapped inside the EJBs. The web designers, like JSP writers, and other EJB clients access the data through EJBs instead of SQL statements. A typical EJB based application would look like *figure 1*.

In the center is the EJB container. It manages a set of enterprise beans. The beans connect with the backend systems, typically a relational database system. The web container typically uses JSP to access the EJB, and serve the JSP to the browser. Besides, EJB client applications other than JSP, such as Java applets or any other systems, can also access to the enterprise beans.

The enterprise beans in the middle tier function as wrappers over various systems, especially relational database systems. There are two kinds of enterprise beans, i.e., entity bean and session bean. Roughly speaking, session beans are the verbs and entity beans are the nouns of the application. An entity bean is a persistent object that represents an item in a storage system such as a database system. In a simple scenario, one bean could correspond to a row in the table. The selection of certain rows of the table corresponds to the selection of a group of beans satisfying certain condition. The selection of beans is accomplished by the finder method in the entity bean, which has SQL statement imbedded in the method. What we are interested in is the entity bean, especially the CMP entity bean. This is illustrated in *figure* 2.

By using EJB, web designers no longer need to learn the details of database structure. Also, any changes in the database are shielded off by the beans, thus making the maintenance of the JSP pages easier.

## 2.2 Object Relational Mapping in EJB

Object-relational mapping is the process of transforming between object and relational models and between the systems that are built on top of these models. While it has been extensively studied as for the mappings between relational and object models, the mappings between the SQL expressions and the behaviors of the objects are barely touched. With the introduction of Enterprise JavaBean (EJB), there are growing demands for supporting the transformation of SQLs in legacy systems to EJB.

A simple object–relational mapping can be illustrated in *figure* 3. In this mapping, the table `Emp` maps to the class `EmployeeBean`, the columns `NAME` and `SAL` corresponds to the attributes `name` and `sal` in the class, respectively. For the query in the relational database such as finding all the employees by name, there is a

corresponding method called `findByName` in the entity bean that has a SQL expression embedded inside the method.

Object-relational mapping is often complicated in several ways. First, object and table may not be a simple 1-1 mapping. Instead, it may be a many-many mapping in many cases. The same applies to the attribute-column mapping. Second, there are relationships between objects, like association, aggregation, and inheritance. In the relational model there are also associations between tables realized by the foreign key. When mapping the relational model to the object model, we need to map those relationships as well. This can be illustrated in *figure* 4.

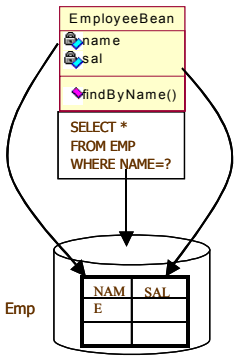


**Fig. 3.** Simple mapping

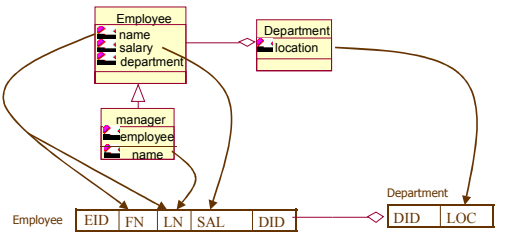


**Fig. 4.** Complex mapping

### 2.3 Query Rewriting

Query rewriting problem has been extensively studied in the areas of query optimization [10]and data integration [25]. Informally speaking, query rewriting problem can be formulated as follows. Given a query and a set of view definitions, how can we answer the query using the answers to the views? Following the common practice we use Datalog notation [35] in the following discussion.

**Definition 2.1** (*Query containment and equivalence*) A query Q is contained in another query Q', denoted as $Q \subseteq Q'$, if for any instance of the base relations, the set of tuples computed for Q is a subset of those computed for Q'. Two queries Q and Q' are equivalent (denoted as Q=Q') if $Q \subseteq Q'$ and $Q \supseteq Q'$.

**Definition 2.2** (*query rewriting*) Given a query Q and a set of views V, a *rewriting* of Q using V is a query Q' such that Q=Q', and Q' refers to one or more views in V.

## 3 EJB Reengineering Framework

Here we define a generic framework to reengineer database applications to EJB based architecture. When reengineering such applications, we face two challenges that are depicted in *figure* 5:

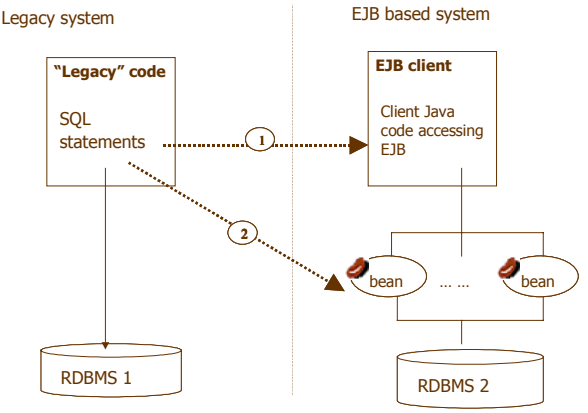


**Fig. 5.** Issues database application reengineering

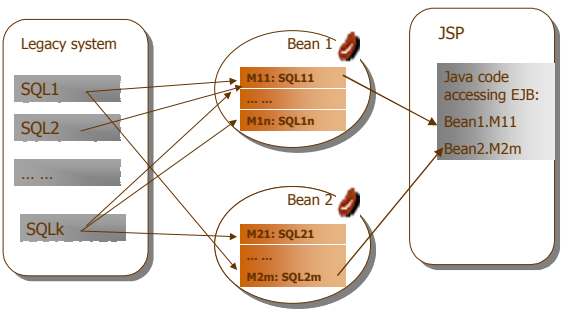


**Fig. 6.** White-box reengineering

1. When the enterprise beans are already provided, how to translate the queries embedded in the legacy code to the equivalent EJB client code?
2. When the enterprise beans are not provided, how to produce the beans, especially the finders and the queries inside finders according to the existing legacy queries?

In general these two tasks are not performed independently. The reengineering is an iterative process with intervention from EJB designers. In a typical scenario the EJB designer defines some entity beans first, then uses the SQL-EJB mediator to locate legacy SQL queries that can not be reproduced using the enterprise beans. Those queries are then used as the basis for EJB generation. By using the EJB generator, enterprise beans and the finder methods are recommended for the designer to choose. The selected beans are added into the existing system and we can iterate this process, run the SQL-EJB mediator once again.

As for the SQL to EJB reengineering, there are two approaches. One is the black-box reengineering. In this approach, the SQLs are not analyzed. Instead, they are directly copied into the methods of enterprise beans. This kind of reengineering requires generating some scaffolding code for the method so that it can access the database. In general, this approach will use session beans.

On the other hand, the white-box reengineering approach that is described in this paper is much more complicated. This will require the generation of entity beans and the rewriting of the queries into object interface. Several queries may be combined into one method in one entity bean, or one query may be split into several methods in different beans, or as illustrated in *figure* 6. This is a good long-term solution offering a clean object-oriented architecture.

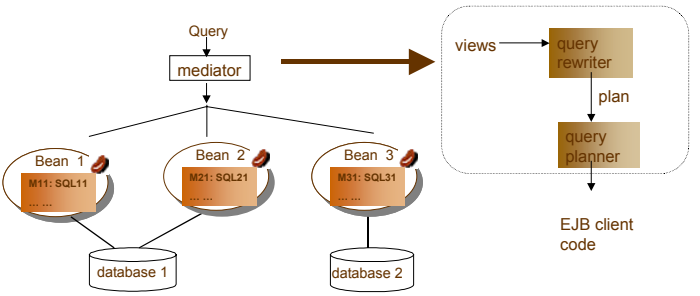


**Fig. 7.** SQL-EJB mediator

# 4 EJB-SQL Mediator

## 4.1 SQL-EJB Mediator Architecture

Given a query in a legacy system, and a set of enterprise beans with SQL statements imbedded in the finder methods in the EJB architecture, the task of SQL-EJB mediator is to identify the enterprise beans, their relevant methods, and the way to combine those methods.

Obviously, there are several questions need to be answered before we can automate the reengineering of the SQL queries to the client Java code in the EJB architecture:

1. For a query in the legacy system, is there a way to decide which finder method in EJB architecture is semantically equivalent to the query?
2. When there does not exist a single corresponding finder method, is there a way to find a bunch of finders that are semantically equivalent?
3. After we decompose the query into several sub-queries, how to combine them in the EJB client Java program?
4. When those sub-queries (finders) do not exist, what kind of finders we need to add? More specifically, what kind of finders we need to add so that it is guaranteed every query in the legacy system can be represented in the EJB?

If we regard the finders as the view definitions in database, we can see that the first question corresponds to query containment [8], the second corresponds to query rewriting using views [26], the third corresponds to query planning and the fourth the view selection [27][6].

The SQL-EJB mediator can be illustrated in *figure* 7, which resembles many of the information mediator systems [25]. The main difference is that we have enterprise beans instead of heterogeneous information sources, and we produce client Java code instead of actually running the queries. Like information sources, enterprise beans are wrappers of the one or more databases and have limited query answering capabilities.

### 4.2 Representing the Query Answering Capabilities of the Enterprise Bean

One factor that makes the query rewriting systems different is how the views are represented. Remember the views are the SQL queries in the finders in the entity beans. Due to the EJB specification, there are several format requirements for the view definition:

- The finder method always returns one entity bean or a collection of beans. That means the head of the view definition will always have attributes from one particular bean. It will never be able to contain attributes from different beans. Also, views always select all the attributes from a entity bean.
- The finders have arguments. That means the view definition would be parameterized.
- The entity bean will have getter for each attribute. That means by default there is a projection for every attribute in the base relation.

Using these observations, we extracted the view definitions from the EJB source code.

### 4.3 Maximal Query Rewriting in SQL-EJB Mediator

When designing a query rewriting system, there are two factors need to consider:

1. *What if there are multiple choices of the rewritings*. In the case of optimizing queries using rewriting [10], the best rewriting could be selected by comparing the cost of each candidate.
2. *What if there is no complete and equivalent rewriting of a query*. A complete rewriting is the one that contains view predicates only. In many cases, a complete and equivalent rewriting of a query using a set of views may not exist. When such a case occurs, there are two choices. One is to relax the completeness requirement and allow view predicates as well as base predicates occur in the rewriting. The other is to compromise the equivalent requirement and make the rewriting not equivalent but the maximally-contained rewriting.

When we design the rewriting system, we have the following considerations:

1. When there are multiple complete rewritings, we select the one that has fewer views. This is because each view will have separate database access. Less views means less database connections and travels. Also, there is less code at the client side to integrate the result
2. When there is not a complete rewriting, we sacrifice the completeness and minimize the number of base predicates. This is necessary since we want to have equivalent rewriting. Besides, the base predicates that can't be removed from the query will be suggested as new view definitions. Those new definitions will become the input of the EJB generator and finally will be added as some finders in some entity beans. We would like to see fewer views the better.

Given the above design desiderata, in the following we formalize those requirements and present the algorithm that satisfies those conditions.

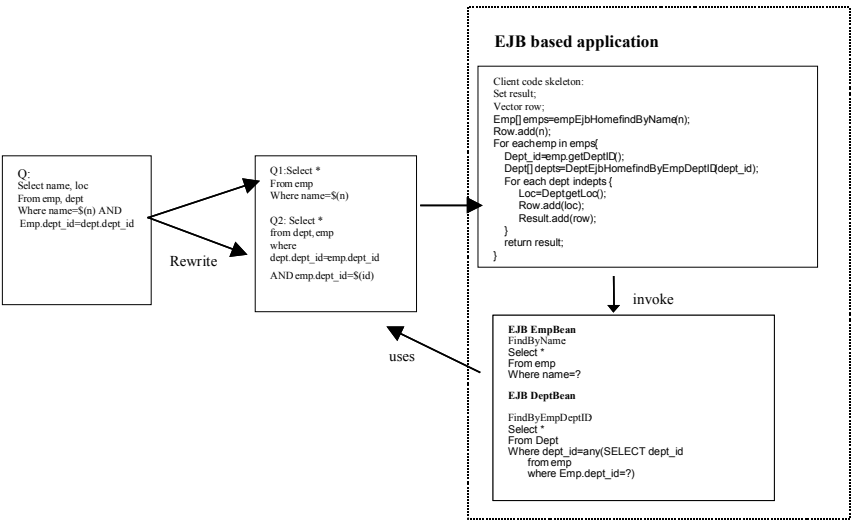


**Fig. 8.** Code generation

**Definition 4.1** (*Partial and complete rewriting*) For a rewriting Q' of a query Q using a set of views V, when the body of Q' contains view predicates from V as well as base predicates, it is called *partial rewriting*. When the body of Q' contains view predicates only, it is called *complete rewriting*.

**Definition 4.2.** (*Maximal rewriting*) Given two partial rewritings Q' and Q'' of Q using views V. Q' is greater than Q'' if the number of base predicates in Q' is less than that of Q''. A partial rewriting Q' is *maximal* if there is no other partial rewriting Q'' of Q using V such that Q'' is greater than Q'.

Please note that the maximal rewriting is different from the maximally-contained rewriting[30].

### 4.4 Extended Bucket Algorithm

There are two classes of rewriting algorithms. One is used in query optimization that guarantees the efficiency and sacrifices the completeness, such as join enumeration [26]. The other is used in data integration that guarantees the completeness and sacrifices the equivalence, such as the Bucket algorithm [26] and Inverse-rule algorithm [26]. Our approach is to extend the Bucket algorithm so that equivalence and the maximalarity as defined in previous section are ensured. Besides, we have other requirements that are specific in our particular application area.

### 4.5 Generate the EJB Client Code

Once a rewriting is obtained, we need to generate a plan for the query so that client Java code could be generated to actually execute the query. A query plan is a

sequence of accesses to the EJB methods interspersed with local processing operations. Given a query Q of the form:

$$Q(X):-V1(X1),\ldots,Vn(Xn).$$

A plan to answer it consists of a set of conjunctive plans. Conjunctive plans are like conjunctive queries except that each subgoal has input and output specification associated with it. For example, a plan for the above query could be

$$Q(X) :- V1(X_1)(In_1,Out_1), V2(X_2)(In_2, Out_2), \ldots \ldots Vn(Xn)(In_n, Out_n).$$

A plan is executable if the input of the i-th predicate appears in the output of the preceding predicates, i.e., $In_i \subseteq Out_1 \cup \ldots \cup Out_{i-1}$

Once we generated the execution plan, we can replace the view predicates with finder methods invocations, and provide the input parameters using input/output definitions.

*Figure* 8 explains a concrete example. The starting point is a SQL query like Q and a bunch of enterprise Java beans like `EmpBean` and `DeptBean`. In those entity beans there are some finder methods like `findByName` and `findByEmpDeptId`, which have their corresponding SQL queries. When page designers want to access database, they will use the beans instead the SQLs. In this case, we need to decompose query Q into Q1 and Q2, which corresponds to `findByName` and `findByEmpDeptID`, respective. Then, according to the logic of the decomposition, we need to insert Java code to combine the results of the two finder methods.

# 5 EJB Architecture Generation

## 5.1 Rational for EJB Generation

The problem is given a set of legacy queries, how to produce the finder methods and SQLs inside the finder methods, so that all legacy queries have a rewriting using the finders?

Obviously, a naïve solution for this problem is to define a session bean for every legacy query. On the other end of extreme, for every base predicate we can define an entity that has finders for every attribute of the predicate. The former approach can be used in the black-box reengineering discussed in Section 3. The drawback is that we will have session beans only and henceforth, it loses the beauty of the entity bean and is not a really EJB system. Besides, queries in EJB have various format requirements. We need to transform the legacy queries anyway to fit in the EJB specification. Also, from the EJB design point of view, enterprise beans are meant for the programming interface for bean users. The interface should be kept simple and logically coherent. That entails the decomposition, classification, and the generation of a minimal set of finders.

The desiderata for the finder selection (or view selection) aiming at EJB architecture generation are as follows, in both semantic and syntactic aspects:

1. *Complete*: every query should have a rewriting using the finders.

2. *Minimal*: The set of queries that can be answered using the finders should not be much larger than the original query set. One of the purposes of adding the EJB layer on top of the database is to provide partial database query capabilities so that the modularity and security can be increased.
3. *Concise:* The number of views (i.e., the EJB finders) should not be very large. Remember that page designers use EJBs as an interface to access the data. The interface should be as succinct as possible. A direct consequence of this requirement is that the size of the views should not be very large. Smaller views can generate more queries.
4. *Efficient:* The views should do the time-consuming operations as much as possible. This is because the views (the finders) are supported by the underlying database, while composing the finders would be done outside of database which is not as efficient as the database.

In the following we formalize those requirements.

**Balloon algorithm**
Input: a set of queries Q, constants $k_1$ and $k_2$.
Output: A set of views V.
Steps:
1. Let V=Q.
2. For every $V_i$ in V, suppose $V_i$ is of the form:
   $V_i$:-$P_{i1}$, $P_{i2}$, …, $P_{im}$.
   Let $V_{i1}$ :- $P_{i1}$. , …, $V_{im}$ :- $P_{im}$.
   Add each $V_{ij}$ to V.
3 For each combination of ($P_{i1}$, …, $P_{im}$) in $V_i$ with length less than $k_1$ do {
   Suppose the combination is $P_{il}$, …, $P_{in}$.
   Let $V_k$=$P_{il}$, …, $P_{in}$.
   Rewriting the queries in Q using V.
   Count the number of times that $V_k$ is used in all the rewritings;
   If count > $k_2$ {
      For each view in {$V_{il}$, …, $V_{in}$} {
        If every query in Q has a rewriting after removing the view
        Then remove the view.
      }
      Add $V_k$ into V.
   }
}

**Fig. 9** Balloon algorithm

### 5.2 The View Selection Problem

EJB architecture generation can be studied in the realm of view selection problem [27][14], which is a dual problem of query rewriting. Informally speaking, it can be formulated as: given a set of queries, how can we find a set of views so that all the queries have a rewriting using this set of views

**Definition 5.1** *(covering view set)*: Let $\mathcal{Q}(V)$ denote the set of queries that have rewritings using V. Given a set of queries Q and a set of views V. V is a covering view set of Q if $Q \subseteq \mathcal{Q}(V)$, i.e., for every query q in Q there is a rewriting of q using V.

Note that in general $Q \subseteq \mathcal{Q}(Q)$.

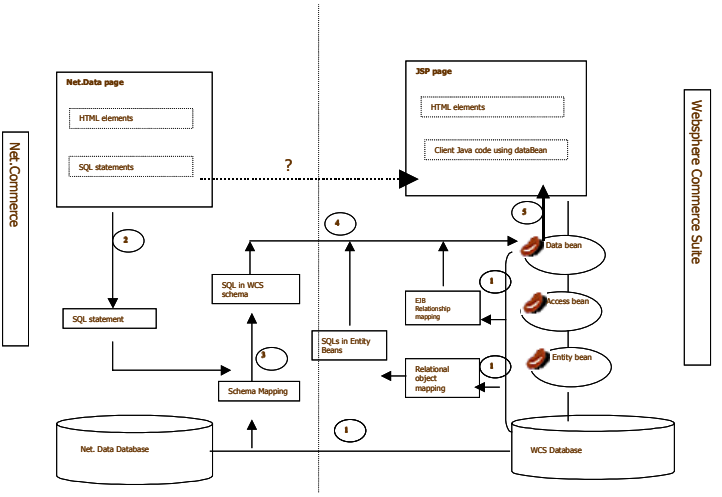


**Fig. 10.** Architecture of the Websphere reengineering tool

**Definition 5.2** (*view set containment*) Given two view sets V1 and V2, V1 is contained in V2 (denoted as V1 ≤ V2) if $\mathbf{Q}$ (V1) ⊆ $\mathbf{Q}$ (V2), i.e., for every query Q, if Q is answerable using V1, then Q is also answerable using V2.

By the definition, it is straightforward that if V1 ⊆ V2, then V1 ≤ V2. But the vice versa does not hold.

The following property says that if we split a view in a view set, the view set becomes larger.

**Property 5.1** For a view set V={V1,…, Vi, …, Vn}. Let Vi:-P1,P2. Vi1:-P1. Vi2:-P2. V'={V1, …, Vi1, Vi2, …, Vn}. Then V ≤ V'.

Note that the view set containment is defined in terms of infinite number of queries. The following property ensures the containment is decidable.

**Property 5.2** Given two view set V and V'. V ≤ V' if for every Vi in V, there is a rewriting of Vi using V'.

Using the view set containment relation, we can define the minimal covering view set.

**Definition 5.3** (*minimal covering view set*) V is a minimal covering view set of Q, if V is a covering view set of Q and there is no other covering view set V' of Q such that V' ≤ V.

### 5.3 Balloon Algorithm

Based on the properties above, we present the balloon algorithm in *Figure* 9. The starting point is the set of queries Q. When we take the Q as the set of views, the queries that can be answered using Q is $\mathbf{Q}$ (Q), which is larger than Q. After that, we blow the balloon to its full extent, get the $\mathbf{Q}$ (split(V)). After that, we let some air out of the balloon $\mathbf{Q}$ (split(V)), but never let it smaller than $\mathbf{Q}$ (V). How large the solution

balloon is will depend on the parameters $k_1$, the size of the views, and $k_2$, the number of the times of a view is used in rewriting all the legacy queries. The larger is the $k_1$, the smaller the balloon would be.

By adjusting $k_1$ and $k_2$, we will be able to get a spectrum of view selections, each one satisfies the semantic and syntactic requirements, and serve as the recommendation for the bean developer to choose.

**Property 5.3** The view generated is complete wrt the query set.

**Property 5.4** The view generated is smaller than $\mathbf{Q}$(split(V)).

**Table 1** Differences between the two systems

| | Net. Commerce | Websphere Commerce Suite |
|---|---|---|
| Presentation language | Net.Data | JSP |
| Business logic | SQL | EJB |
| Programming language | C++ | Java |
| Programming model | Ad hoc | MVC |

## 6 A Case Study: Reengineering the IBM e-Commerce Framework

To validate the usability and feasibility of the approach discussed above, we have been carrying out an e-commerce reengineering project for IBM Toronto Lab. The project aims at the transition from the 5-year-old e-commerce framework Net.Commerce to the new architecture Websphere Commerce Suite that uses EJB technology.

Both Net.Commerce and Websphere Commerce Suite are frameworks for building e-commerce websites. They provide templates for realizing functionalities such as product catalog browsing, payment processing, product promotion, auction, and almost anything that can be carried out on the web. Both of them are multi-tier applications that use browsers at the client side, use web-servers in the middle, and use the RDBMS to manage the data.

Although there are many similarities, they are actually totally different systems. WCS is a successor of Net.Commerce. It is redesigned without the consideration for backward compatibility. Most notably, WCS uses EJB technology. Table 1 highlights the differences between the two systems. *Figure* 10 illustrates simplified architectural views of the two systems.

For the presentation language, Net.Commerce uses Net.Data, which is an IBM proprietary product for dynamic HTML, while WCS uses JSP (the Java Server Page) that is widely accepted for dynamic web pages. For the business logic, Net.Commerce uses SQL in many cases directly in C++ and Net.Data, while WCS uses EJB to access the database. For the programming language, Net.Commerce uses C++, while WCS uses Java. For the programming model, Net.Commerce is pretty ad hoc. Sometimes the separation of view, model, and control is not clear. WCS adopts a clean MVC (Model, View and Control) design pattern.

Given the huge size and complexity of the two frameworks, there are many reengineering tasks. Our focus is on the Net.Data to JSP reengineering, and especially, the SQL to EJB reengineering. *Figure* 10 shows the overall task and the steps we take. On the left side is the Net.Commerce architecture, right side is the WCS architecture. This picture is oversimplified to focus on the things we are interested in. Net.Commerce uses Net.Data to generate dynamic web pages. In Net.Data, there are SQL statements that can access databases. In WCS, JSP is used as the presentation language. In JSP there are Java code that accesses data beans. Data beans and access beans are IBM extensions to EJB framework to improve performance.

Our first step in the reengineering is to analyze the WCS source code to extricate several relationships. One is that between the data beans, access beans, and enterprise beans. The other is the relationship between the data beans such as inheritance and aggregation. The third is the relationship between the objects and relational database, such as the mappings between the objects and tables, between the attributes and columns, and between the methods and SQLS. In the meanwhile, we analyze the mapping between the two relational databases in the Net.Commerce and WCS.

The next step is to parse the Net.Data and the SQLs inside Net.Data. Using the mapping information extracted in the first step, the third step is to translate the SQL in the old schema to the equivalent one in the new schema. The fourth step is to compare the legacy SQL with the SQLs in the WCS and pick out the most relevant ones. Using the relationships between SQLs and objects and that of different objects, step 5 make the recommendations that the enterprise beans we should use in the JSP page.

We have implemented the system and have published it in IBM alpha works [24]. In addition to that carried out on the Websphere customer side, the tool has been tested on more than 500 Net.Data file, more than 600 SQLs. The largest Net.Data file is 152K. The WCS source code that we crawled through is 62MB. Altogether there are more than 200 tables and more than 200 entity bean. The view set collected from the WCS source code consists of about 600 of queries.

This e-commerce reengineering project shows that the white-box reengineering, especially the SQL-EJB mediator and the EJB architecture generator, is crucial for the success of EJB application development. Also, this project demonstrates the feasibility of our approach. Through this project, we also realized that EJB design should rely on not only the logics in queries, but also on the design strategies that have been evolved over time and experience, such as what granularity of the entity beans should have, when to use BMP or CMP entity bean, and when to use session bean. Those EJB design strategies need to be studied, formalized, and applied in the EJB generation.

## 7 Related Work and Conclusions

There are tools and methods to migrate EJB applications from one platform to another [1][20]. Our work differs from those approaches in that we are migrating the database applications to EJB architecture instead of moving EJB applications between different platforms. There are also tools for mapping database schemata or UML models to

entity beans[18][17]. However, these do not address the SQL issues. Such tools are more relevant to the traditional object-relational mapping problem [2][12].

Database reverse engineering and schema mapping [21][36] are relevant to our work as well. Most database reverse engineering research attempts to map a relational schema to an object schema, or the transformation of relational queries to object-oriented queries [9][12]. In our case, we translate SQL expressions from database application to object wrappers of the relational database.

Compared with the work in query rewriting and view selection, we have several contributions as well. First, we proposed a new applications area for query rewriting and view selection which entails a different approach for query rewriting and view selection. For the query rewriting, we developed the notion of maximal rewriting and the extended bucket algorithm. The SQL-EJB mediator differs from other information mediators in several respects, such as the query capability representation in EJB and the query rewriting algorithm. More importantly, this is the largest query rewriting system ever used in real application. Up to now, the largest query rewriting system we know of is described in [30], which experimented with hundreds of views. However, the views and queries there are randomly generated. In our case, thousands of queries and hundreds of views come from real industry application. Thirdly, unlike other query rewriting system, we translate query plans to Java programs. For the view selection, we proposed the balloon algorithm in order to generate the EJB architecture.

The research on object-relational mapping has been largely on the schema level, i.e., between object and relational models. There has been few work on the study of the mappings between the systems that are built on top of these models, and in particular, the mapping and translation between the SQLs and the methods in the objects. In the settings of Enterprise JavaBean technology and software reengineering application area, this paper demonstrates the importance of such a mapping and presents the methods to do the translation. We should emphasize that our methods is applicable not only to the EJB reengineering problem, but also to other object persistent mechanism, such as JDO (Java Data Object).

From the reengineering point of view, we propose the reengineering of database applications using query rewriting and view selection techniques.

This is an on going project. There are several issues need to be investigated further.

Although a reengineering tool is constructed and commercialized, and the approach proposed in this paper is proved necessary and feasible in this reengineering environment, the implementation of the SQL-EJB mediator and EJB generator is still under way.

In this paper the EJB architecture is assumed to be a simple one that does not have inheritance and associations [17]. Also, we are only considering CMP entity beans. BMP entity beans and session beans are not considered.

When rewriting a query the cost model is not used for selecting a good rewriting over a bad one.

When selecting views from a set of queries, now each query is considered of equal importance. However, a more precise approach is to use the workload model, which is a set of queries and each query has a weight that reflects how often the query is used.

All our algorithms are assuming SPJ queries. How to deal with disjunctive queries, groupings etc needs to be investigated.

In the new version of EJB specification EJB 2.0, a vendor independent query language EJB-QL is defined. We need to expand our approach to cover the EJB-QL, instead of SQL.

## Acknowledgements

The author would like to thank John Mylopoulos for the stimulating discussions of the problem and his valuable comments, modifications on this paper, also Kostas Kontogiannis, Terry Lau, Emily Xing and Erik Hedges for their input in the construction of the IBM E-commerce reengineering tool.

## References

[1] Sanjay Agrawal, Surajit Chaudhuri, and Vivek Narasayya. Automated selection of materialized views and indexes in Microsoft SQL Server. VLDB 2000.

[2] R.S. Arnold, editor. *Software Reengineering*, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[3] Andreas Behm and Andreas Geppert and Klaus R. Dittrich, *On the Migration of Relational Schemas and Data to Object-Oriented Database Systems*, in Proc. 5th International Conference on Re-Technologies for Information Systems, 13-33, 1997.

[4] S. Bergamaschi and A. Garuti and C. Sartori and A. Venuta, *The object wrapper: an object oriented interface for relational databases*, In Euromicro 1997.

[5] Kyle Brown, Handling N-ary relationships in VisualAge for Java, http://www.ibm.com/vadd, August 2000.

[6] Elliot Chikofsky and James Cross. *Reverse Engineering and Design Recovery: A Taxonomy*. IEEE Software, 7(1): 13-17, January 1990.

[7] Rada Chirkova, Alon Y. Halevy, Dan Suciu, A Formal Perspective on the View Selection Problem, VLDB 2001.

[8] A.K. Chandra, P.M. Merlin, *Optimal implementation of conjunctive queries in relational databases,* in Proceedings of the 9th Annual ACM Symposium on Theory of Computing, pages 77-90, 1977.

[9] Chang, Y., Raschid, L. and Dorr, B., *Transforming queries from a relational schema to an equivalent object schema: a prototype based on F-logic*, Proceedings of the International Symposium on Methodologies for Intelligent Systems, 1994.

[10] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Pptamianos, Kyuseok Shim, *Optimizing Queries with materialized views,* ICDE 1995.

[11] Cohen, Y.; Feldman, Y.A., *Automatic high-quality reengineering of database programs by temporal abstraction,* Proceedings of the 1997 International Conference on Automated Software Engineering (ASE '97) (formerly: KBSE)

[12] Fong, J., *Converting Relational to Object-Oriented Databases*. SIGMOD Record, Vol.26, No. 1, March 1997.

[13] H. Gupta, I. S. Mumick, *Selection of views to materialize under a maintenance cost constraint*, in Proceedings of ICDT, pages 453-470, 1999.

[14] H. Gupta, *Selection of views to materialize in a data warehouse*, in Proceedings of ICDT, pages 98-112, 1997.
[15] IBM, *IBM Net.Data Reference*, Version 7, http://www4.ibm.com/software/data/net.data/, June 2001 Edition.
[16] IBM, Websphere Commerce Suite Version 5.1: An introduction to the programming model, IBM white paper, Feb 2001.
[17] IBM, VisualAge for Java 3.5, IBM, 2001.
[18] In2j, Automated tool for migrating Oracle PL/SQL into Java, www.in2j.com, April, 2001.
[19] Ivar Jacobson , Fredrik Lindström, *Reengineering of old systems to an object-oriented architecture,* OOPSLA 1991, ACM SIGPLAN Notices, Volume 26 Issue 11.
[20] IPlanet, *Migration Guide, iPlanet Application Server,* Version 6.0, www.iplanet.com, May 2000.
[21] J. Jahnke and W. Schafer and A. Zundorf, A Design Environment for Migrating Relational to Object Oriented Database Systems, In Proceedings of the International Conference on Software Maintenance, IEEE Computer Society Press, 163--170, 1996.
[22] Yannis Kotidis, Nick Roussopoulos, DynaMat: A Dynamic View Management System for Data Warehouses, SIGMOD 99, June.
[23] Terry Lau, Jianguo Lu, Erik Hedges, Emily Xing, Migrating E-commerce Database Applications to an Enterprise Java Environment, CASCON'01.
[24] Terry Lau, Jianguo Lu, John Mylopoulos, Erik Hedges, Kostas Kontogiannis, Emily Xing, and Mark Crowley, *Net.Data to JSP Helper*, IBM alphaWorks, www.alphaworks.ibm.com/tech/netdatatojsp, 2001.
[25] Alon Levy, Anand Rajaraman, Joann J. Ordille, *Querying heterogeneous information sources using source descriptions*. In proceedings of the international conference on Very Large Data Bases, Bombay, India, 1996.
[26] Alon Levy, Answering queries using views: a survey, VLDB Journal 2001.
[27] Chen Li, Mayank Bawa, Jeffrey D. Ullman, Minimizing view sets without losing query-answering power, ICDT'01.
[28] R. J. Miller, L. M. Haas and M. Hernández. *Schema Mapping as Query Discovery.* VLDB 2000.
[29] Wie Ming Lim and John Harrison, *An Integrated Database Reengineering Architecture - A Generic Approach,* Proceedings of the 1996 Australian Software Engineering Conference (ASWEC '96).
[30] Rachel Pottinger, Alon Y. Levy, A Scalable Algorithm for Answering Queries Using Views**,** VLDB 2000.
[31] William J. Premerlani, Michael R. Blaha, An approach for reverse engineering of relational databases, CACM, 1994 Vol 37(5).
[32] Chandrashekar Ramanathan, *Providing Object-Oriented Access To Existing Relational Databases,* PhD dissertation, Mississippi State University, 1997.
[33] Sun, Enterprise JavaBeans 2.0 Specification, www.java.sun.com, 2001.
[34] Tech Metrix*, Moving from IBM Websphere 3 to BEA WebLogic Server 5.1*, White Paper, TechMetrix Research, September 2000.
[35] Jeffrey D. Ullman*, Principles of Database and Knowledge-base Systems,* Volumes I, II, Computer Science Press, Rockville MD, 1989.