A Methodology for Database Reengineering to Web Services

Ignacio García-Rodríguez de Guzmán, Macario Polo, and Mario Piattini

ALARCOS Research Group
Information Systems and Technologies Department
UCLM-Soluziona Research and Development Institute
University of Castilla-La Mancha
Paseo de la Universidad, 4 – 13071 Ciudad Real, Spain
{Ignacio.GRodriguez, Macario.Polo, Mario.Piattini}@uclm.es

Abstract. Databases are one of the most important components of information systems, since they keep all the information of organizations. Although new standards in databases have appeared in the last years, most databases are still based on SQL-92, and are thus true legacy systems. Most of the services offered by information systems are based on the information stored in their databases. In order to allow interoperability, current trends advise exposing some of these services to the Web, making them available for other users and also for the information system itself. Since dealing with old databases and their associated software is difficult, a methodology to discover services from SQL-92 databases and to offer them via Web Services is proposed. This methodology is based on the MDA approach and implements a reengineering process, which starts from an SQL-92 database and obtains a set of services that can be exposed as Web Services.

Keywords: reengineering, reverse engineering, metamodel, Web Service, QVT, patterns, MDA.

1 Introduction

Information systems are composed of many elements, for example documentation, programs, hardware, databases, etc. Of these, the database can be considered as the cornerstone. This importance is due to the role played by databases: they store all the information required for system operation.

Despite the fact that new versions of the SQL standard are being developed (i.e. SQL-99, SQL-2003), many systems are still working with relational databases [1], mainly based on the SQL-92 paradigm [2] .

Programs that use legacy databases are sometimes also legacy programs with low maintainability. Because of that, the effort spent on improving these systems (adding new features, integration into the web, etc.) can be taxing [3] .

Any attempt to deal with this kind of legacy system is difficult for many reasons, such as the size of both the applications and the databases [4] , lack of experience in the source code language, lack of documentation, etc.

A. Rensink and J. Warmer (Eds.): ECMDA-FA 2006, LNCS 4066, pp. 226 – 240, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Reengineering is a very useful tool for dealing with this kind of problem. According to [5] , reengineering is a process composed of two sub-stages: reverse and forward engineering.

Recently, reverse engineering (the most important stage of reengineering) has become closely related to MDA [6] . In just a few words, MDA makes is possible to separate business logic from the implementation platform [6] . MDA proposes to work at both model and metamodel levels: thus, the implementation stage is not as critical as in the earlier times, because this step can be performed by means of automatic transformations. The relation of reverse engineering and MDA has been strengthened by ADM (*Architecture-Driven Modernization*), which aims to integrate reverse engineering and MDA. Many metamodels have been standardized to support legacy systems, such as CWM [7] . It is essential to represent this kind of systems by means of these metamodels by a reverse engineering stage.

According to [6, 8] , the basic elements of the MDA approach are PIMs (Platform Independent Models), PSMs (Platforms Specific Models), and PDMs (Platform Description Models). When MDA is applied, the software engineer works at a business level with one or more PIMs. Later, and by means of some transformations, one or more PSMs are generated depending on the target platform. If the starting point is not a PIM but a PSM (the legacy system), the process involves two transformations, one to obtain a PIM (representation of the legacy platform) and a second transformation to obtain the target PSM from the PIM [8] . The latter situation is what leads to the aforementioned term “Architecture-Driven Modernization”. In this situation, reverse engineering is required to pass from the starting PSM to the PIM: therefore, reverse engineering is a core element in the application of the MDA approach.

In this respect, a methodology based on the idea of reengineering and focusing on databases, using the concepts of MDA and ADM has been developed. The starting PSM is the SQL-92 database; the different PIMs are the set of metamodels used during the process; while the target PSM is the final set of Web Services to be generated.

The methodology also takes into account the fact that, as happens in many applications, the domain layer used is a reflection of the structure of the database which supports the information managed by the application. In other words, having a multi-tier application [9] , the domain (or business) tier is chiefly responsible for implementing the operations required to achieve the objective for which the system was developed. Because of this, the database can be used not only to extract the static structure in a reengineering process, but also to infer many of the original system functionalities.

The development of a general and partially-automated reengineering process for relational databases requires specifying which kind of relational databases are involved. As far as we know, despite the fact that SQL-2003 is the current standard, most databases are still defined in SQL-92 (or the corresponding subset of SQL-2003). For this reason, we are now mainly focused on obtaining all the characteristics of SQL-92 based databases. By means of a set of inference patterns, it is possible to find potential services in the schema of the database, using a model-driven pattern matching process as a sub-step of our general reengineering process.

This paper is organized as follows: Section 2 provides a brief description of the related work; Section 3 overviews our proposal; Section 4 summarizes the reengineering task to be performed until the services are discovered; Sections 5 and 6 depicts the service discovery; Section 7 deals with service implementation, Section 8 puts forth some conclusions and possibilities for future work.

2 Background

Until recently, data reengineering (and more specifically data reverse engineering) had not been one of the most important topics in reengineering for two straightforward reasons: (1) the traditional partition of software engineering and database systems, and (2) source code reverse engineering seemed more interesting in many aspects in the academic environment [10] . Database reverse engineering can be performed for the following purposes [11-13] : redocumentation, model migration, restructuring, maintainance or improvement, tentative requirements, software assessment, integration, conversion of legacy data, and assessment of the state-of-the-art.

Recovering metadata from databases is a very important issue, because our process starts with a database from which no documentation is available. Here, much research has been done on algorithms and techniques to recover metadata stored in database catalogs. In [14, 15] the authors studied algorithms to extract information about the structure of relational databases. In [16] J.L. Hainaut also made a deep study of the database reverse engineering field.

[17, 18] present a reengineering database project named DB-MAIN. DB-MAIN is a generic methodology supported by a tool of the same name, with the following steps: (1) database structure extraction, and (2) data structure conceptualization.

The MIDAS framework [11] tackles the migration of databases, specifically from net databases to relational databases. This framework also allows for the replacement of database access subroutines by SQL code.

In the field of migration, not all the research tries to tailor the original database to the new model in order to adapt it to a new technology. *Wrapping* can be seen as another kind of migration, in which a logical layer is displayed between the databases and the system. This technique used to be implemented by means of *wrappers* which can be seen as a kind of component, such as an adapter. Wrappers make it possible to transform queries for a particular data model into another one, for example from a particular DMS (*Data Management System*) into a different model [19] . Wrappers are used not only to adapt one data model to another, but also for other purposes, for example adapting a relational database to a distributed environment [20] . In this case, the wrappers work as a *façade* between the database and any external system which attempts to access the information.

However, databases are not always subjects of transformations or migration in reengineering, that is, of operations that (sometimes) modify their structure and require complex data transformations. In [21] , the authors implement a whole

reengineering process in a tool, *RelationalWeb*. This tool takes a relational database as input (nowadays it accepts four sorts of DBMS, namely Microsoft Access, SQL Server, Caché Intersystems and Oracle) and generates a full operational application to manage it.

Traditionally, research in database reengineering has focused on the tasks discussed above (migration, restructuring, etc.), but not much research has been done (to our best knowledge) on generating services from relational databases. As with [21], our intention is not to generate applications but rather Web Services, offering operations based on the structure of the database.

Our methodology starts with a particular sort of relational database (as noted above), SQL-92. This is due to the fact that industrial studies show that many information systems are still running over relational databases [1, 22] .

3 An Overview of the Methodology

Fig. 1 illustrates the different steps of the methodology. The first step reverse engineers the SQL-92 database in order to obtain its structure. It obtains an instance of a relational database metamodel representing its complete logical schema.

In the next step, the instance (a model) of the database metamodel is transformed into an instance (also a model) of a metamodel describing an object-oriented representation. This model is instrumented with basic operations and state machines to define its behavior.

Then, the engineer guides the process of service discovery, applying different techniques. As a result, a set of services are shaped in an abstract manner.

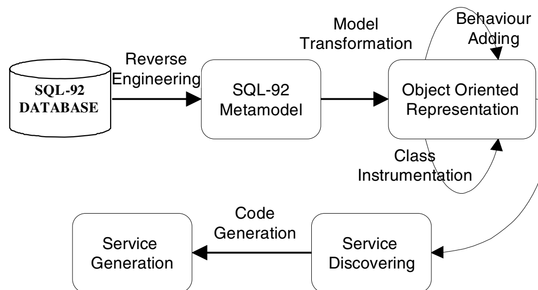

Fig. 1. Process overview

The last stage is the service generation. By means of transformations, a code implementation of the abstract service description is generated. All the stages are explained in the following sections.

4 Preparing the Environment: A Reverse Engineering Task

This section explains the different steps of the methodology in detail.

4.1 Database Metadata Extraction for Schema Representation

An effective strategy for recovering the database schema is used in [23, 24], where database metadata is extracted via specific queries thrown against the database data dictionary.

All these metadata are stored and represented via the SQL-92 metamodel in Fig. 2, proposed by [25] , which considers all the elements of the SQL-92 standard [2]. By means of this metamodel, all the metadata stored in the original database can be represented. From now on we call this sub-metamodel SQL92Schema.

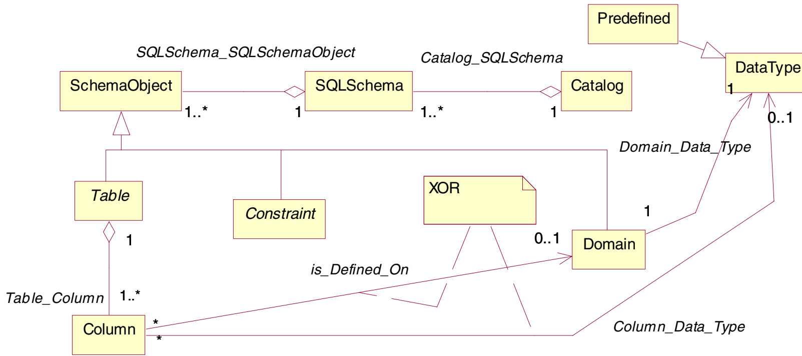

Fig. 2. General View of the SQL-92 metamodel

4.2 An Object-Oriented Representation of the Database

Once the structure of the database has been recovered, the next step in the process translates the instance of the database metamodel (SQL92Schema) into an instance of an object-oriented metamodel, representing the conceptual schema corresponding to the database. This object-oriented representation is the starting point for inferring and building services, establishing a layer between the exposed services and the database itself.

The object-oriented metamodel is extracted from the UML 2 specification [26]. This metamodel has been made up by a subset of the *Classes* package of UML2, but has not been included here due to the lack of space.

4.3 Transforming the SQL92Schema into the OOSv2 Metamodel

This section explains how to obtain an object-oriented representation from a database schema by means of a QVT composed transformation.

All the elements managed are models: thus, the reengineering process is platform-independent and the database service discovery is performed in a conceptual level. In this way, the generation of source code is deferred to the final stage, when the artifacts implementing the services must be generated.

The QVT language [27] was chosen to perform the transformations. QVT is a powerful language to specify transformations among models (and in the same fashion

metamodels). QVT includes both a syntactical and graphical notation to define transformations.

Due to the extension of the complete transformation, Table 1 shows only a small part of the QVT algorithm to transform an instance of the SQL92Schema metamodel into an instance of the OOSv2 metamodel.

Table 1. QVT transformation to obtain an object-oriented system from an SQL-92 Schema

transformation SQLSchemaToOOSv2 (sql92db: SQL92Schema, oos2: OOSv2){ **key** Class**{**name, owner**};** **key** Association**{**name, owner**};** **key** Property**{**name, owner**};** **top relation** SQL92SchemaToOOSv2{…} **top relation** TableToUMLElement{…} **top relation** ConstraintToUMLElement{…} **relation** ReferentialConstraintToAssociation{…} **relation** UniqueConstraintToProperty{…} **relation** BaseTableToClass{…} **relation** ColumnToProperty{…}	**relation DomainToUMLConstraint**{…} **relation ViewToClass**{…} **relation AssertToConstraint**{…} **relation TableCheckConstraintToUMLConstraint**{…} //Funciones **function** SQL92_ValueToOOSv2V_Value (domain sql92 col:Column{}): domain oos2 val:ValueSpecification{} **function** SQL92_TypeToOOSv2_Type (domain sql92 type: DataType{}): domain oos2 type:DataType{} **function** DomConstraintToUMLClassInvariant (domain sqp92 cons:Constraint{}): domain oos2 cons:Constraint{}

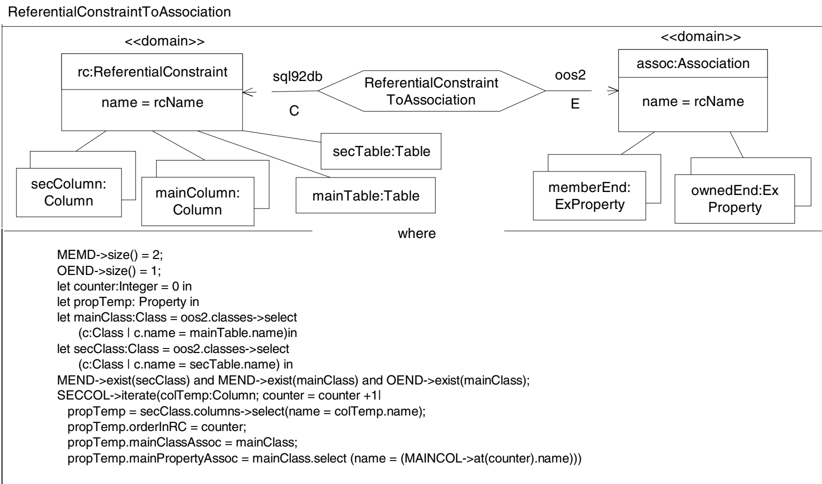

Fig. 3. QVT transformation to obtain a UML association from an SQL-92 foreign key

The algorithm here is a composed function that triggers the functions shown in Figs. 3 and 4 (graphical representation): for example, *View2Class*, *ReferentialConstraint2Association, BaseTable2Class, Assertion2UMLConstraint*, etc. In the same way, other transformations are triggered later, as a consequence of the execution of the main transformation (Table 1).

Transformation in Fig. 4 is in charge of transforming a column from a table to a property of a class. In the same way, we have developed another transformation

which is also invoked by the *SQL92S_2_OOSv2*, the *ReferentialConstraint2-Association* transformation in Fig. 3. This takes a foreign key (which is composed of one set of referencing columns and one of referenced columns) and generates a UML Association (between one class representing the referencing table and other class representing the referenced table).

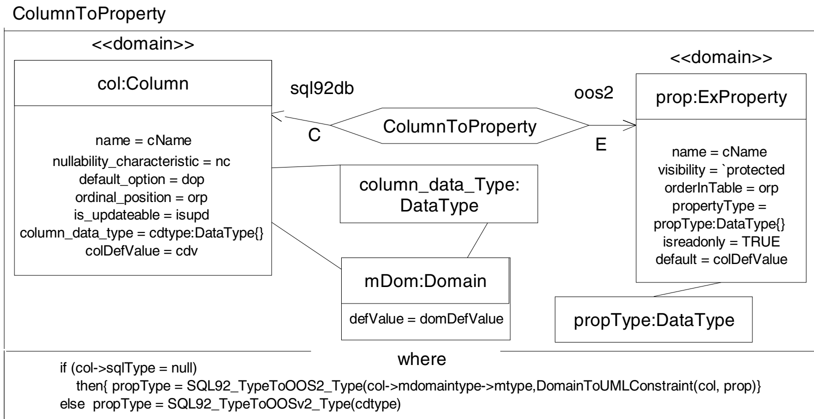

Fig. 4. QVT transformation to obtain a UML property from an SQL Column

5 Class Instrumentation

At this stage of the process, all the work is focused on the instance of the OOSv2 metamodel (that is, the representation of the relational database schema). That means that all classes involved in a service are in fact an object-oriented representation of the relational database tables and the parameters of a service are required to perform an operation over the database.

Before performing any task for service discovery, the set of obtained classes must be instrumented with methods. Up to now classes have been created from tables, and each class owns properties but no methods. Thus, before composing any service, methods must be assigned to all classes.

The basic CRUD operations (***C***reate, ***R***ead***,*** ***U***pdate and ***D***elete, shown in Fig. 5) are added to all classes.

In addition, other operations may be required to add extra behavior to our classes. Imagine the class *Account* representing a banking account (previously recovered from a table). It is very likely that this class would require operations such as *deposit*, *withdraw* or *getBalance*. In other words, many classes would require operations to shape their behavior.

In addition to these operations, a state machine for some classes is also provided in order to give the class a more similar behavior than it has in reality. See [21] for further explanations.

The set of states corresponding to each class can be assigned by hand or be inferred from the data saved in the database. Transitions, however, must always be designed by the engineer, since there is no information about them.

CRUD OPERATIONS

create create (**void**) create (**in**:PK) create (**in**:PK, **in**:custom_ record) **read** read (**in**:PK, **out**:record) read (**in**:PK, **out**:custom_record) read (**in**:custom_PK, **in**:custom_record, **out**:record*)	**update** update (**in**:PK, **in**:{}{\values}) update (**in**:custom_PK, **in**:custom_record, **in**:{}) **delete** delete(**in**:PK) delete(**in**:custom_PK) PK: primary key custom_PK: partial PK, used to select a set of records record: set of columns to be assigned to a table

Fig. 5. CRUD operations to be added

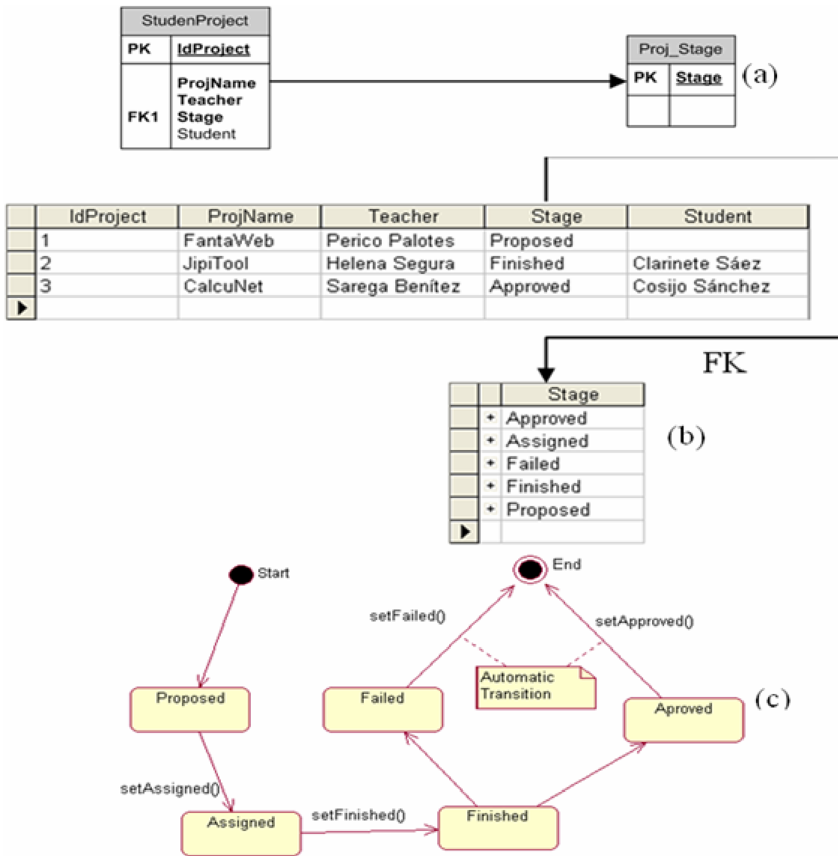

Fig. 6. Obtaining a state machine from a FK

The automation of state discovery inside classes relies on some heuristics. Some of these rules are the following:

- *Use of limit values*: in numeric columns (later transformed to properties), limit values are suggested in order to identify intervals that, at the end, can be seen as

states. For example, remember our *Account* table. The *Balance* column (representing the money in the banking account) must be of a numeric data type. Three limit values can be obtained, namely negative values, zero and positive values. According to this, any account would be in any of these intervals: *Balance-Negative* ($-\infty$ < Balance <0), *BalanceZero* (Balance = 0) and *BalancePositive* (0 < Balance < ∞).

- Now suppose a table, and one of its columns referencing via a foreign key another table with just one column (with a short set of values, such as an enumeration). It is possible that these values are defining the state of the corresponding table record and class instance. For example, in Fig. 6 (a) two tables are observed, the first one representing projects developed by students and the second representing the possible stages a project may have: thus, the *stage* column in the first table is constrained by the second table via referential constraint (Fig. 6 (b)). Taking the values of the *Proj_Stage* table as possible states, a suitable state machine results from this supposition (Fig. 6 (c)).

In order to integrate the class instrumentation inside the full process, the State Machine metamodel of the UML2 specification has been taken into account.

6 Extracting Services for SQL-92 Databases

6.1 Service Extraction

A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services [28] . Also, a service can be seen as an operation which may need a set of parameters and that may return a result.

One of the first issues before generating services automatically is how to build these services in a generic manner. Since a metamodel is an abstract language for some kind of metadata [29] , a service metamodel has been developed to represent any database-based service.

A service can be divided into two basic parts: the static and the dynamic component. The static component of the metamodel represents all the artifacts involved in the service such as classes, parameters, etc. The dynamic component of the service is related to the behavior of the service, that is, the execution flow of the service. This dynamic component lets the engineer model the set of steps that must be taken to achieve the goal of the service.

In the methodology, a service will be translated into an (more or less) complex SQL sentence. That is, both the result of the Model-Driven Pattern Matching (see section 6.2) and the CRUD operations (see section 5).

Due to the SQL representation of services, a dynamic description is not required, but a complete one of the structural elements of the service is essential. With this aim in mind, a service metamodel has been developed. This metamodel does not cover the full syntax of the SQL-92 standard but the required the methodology. Instead of the BNF description of SQL-92 standard this metamodel has been build because (1) it is easier to manage and integrate in the MDA process, (2) also to have a service representation compatible with the other metamodels involved in the process, (3) because this metamodel contains all the required elements to generate the requited

source code to implement services in the code generation stage, and is more suitable than the SQL-92 BNF notation to represent services.

Together with this metamodel, a complete OCL set of invariants has been added to many of the classes in order to specify with are the correct models (of services) that could be generated having the kind of service and the operations involved in the service. Due to the lack of space, these OCL invariants will be studied in further publications.

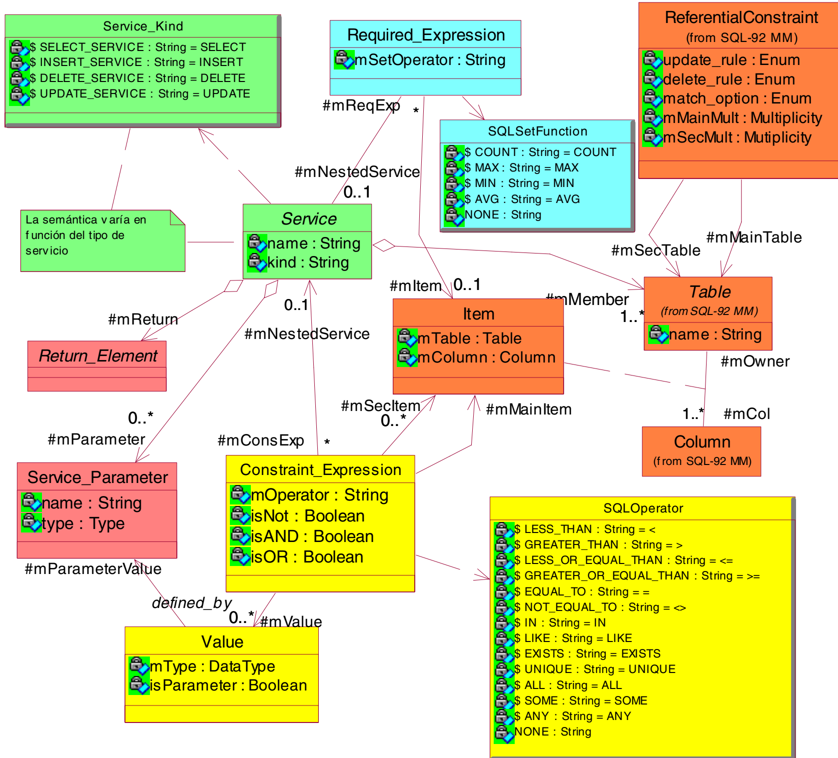

Fig. 7. Service component metamodel

6.2 Model-Driven Pattern Matching

As noted above, the database schema is considered as the reflection of the domain layer, so it is very possible that many of the functionalities of the application are reflected in the schema of the relational databases.

Being M_A the set of elements that could take part in a service, it is possible to match this model against a bigger model, namely M_B, in order to find occurrences of M_A. That is, M_A may be used as a pattern to search inside M_B.

In our context, M_B could be either the SQL92Schema or the OOSv2. The goal of this process would be to obtain a set with all the occurrences of the elements of the model that matches the given specification, and choose among them those that fit our intention.

The idea explained above corresponds to the *model-driven pattern matching* (from now on, MDPEM) concept. This idea is briefly outlined in the MDA specification [6], but emphasized more in the QVT specification [27]: "*The essential idea behind pattern matching is to allow the succinct expression of complex constraints on an input data type; data which matches the pattern is then picked out and returned to the invoker*".

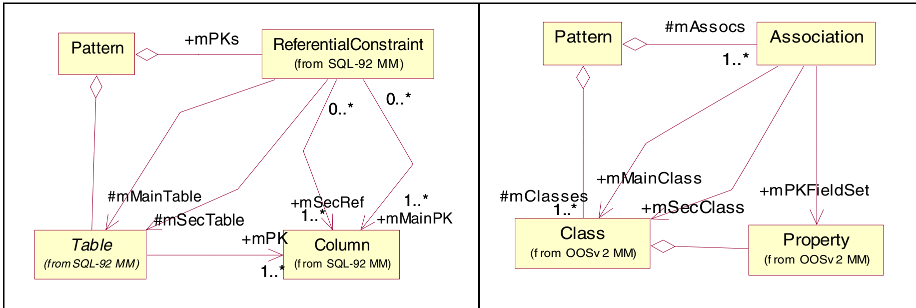

Fig. 8. Pattern metamodels

In this context, the pattern (or the mechanism to shape something that could be offered as a service) is a generic description of a pattern to do the matching against a model. In this respect, the metamodels in Fig. 8 are proposed. These metamodels does not work in the same way as the one proposed in [30], which is more general.

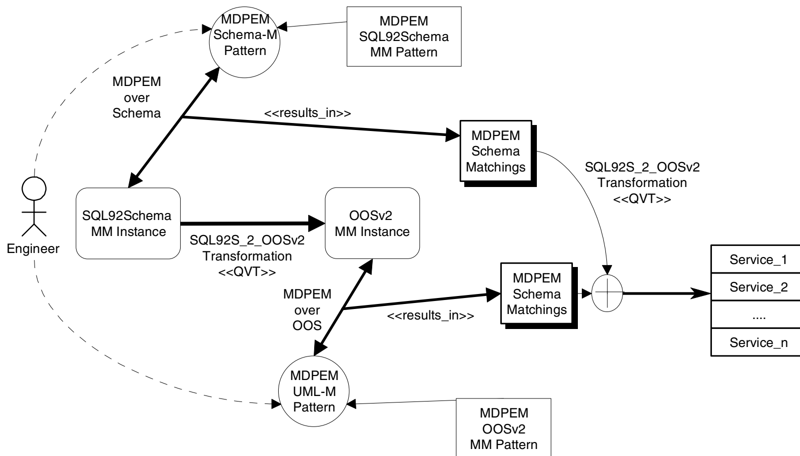

Fig. 9. MDPEM: inputs and results

Because MDPEM can be performed at two different levels of abstraction (namely the SQL92Schema and the OOSv2 level), a pattern metamodel has been designed for each one. While Fig. 8 (a) depicts the metamodel for building a pattern at a OOSv2 level, Fig. 8 (b) depicts a metamodel for builing patterns at the SQL92Schema level.

Looking at the metamodels, it can be seen that both of them represent the same concepts, but at the two different levels of abstraction. This fact means that the MDPEM can be performed over the two models without distinction. Since there is a QVT transformation to obtain an OOSv2 instance from a SQL92Schema instance (see Section 0), this mechanism can be also applied to transform results from MDPEM at the SQL92Schema level to OOSv2 results. This means that the engineer can choose to apply MDPEM to any of the abovementioned levels.

Fig. 9 helps explain the MDPEM process. As Fig. 9 shows, MDPEM is guided by the engineer in charge of the reengineering process. Having both the instance of the SQL-92 metamodel and the object oriented system metamodel, the engineer can choose where to apply the MDPEM technique, depending on the skill of the engineer and his/her preferences, because at the end, all matchings found in the SQL92Schema instance could be transformed to OOSv2 ones by means of our QVT transformations. An example of this process is presented below.

6.2.1 An Example of the Application of MDPEM

Given the relational database in Fig. 10 (b), a pair of patterns (Fig. 10 (a)), which will be applied later, are going to be proposed, both over the SQL92Schema and OOSv2 metamodel instances.

The pattern of Fig. 10 expresses four tables (namely C_p, D_p, E_p and M_p) and three foreign keys (namely $FK_{p1,}$ FK_{p2} and FK_{p3}), where FK_{p1} is a foreign key from M_p to C_p, FK_{p2} is a foreign key from M_p to D_p, and Fkp_3 is a foreign key from M_p to E_p. These elements belong to a conceptual set, namely S, which represents all the elements of the recovered schema. The MDPEM process could be also be expressed by means of a pseudo-SQL query:

$$SELECT\ C_S, D_S, E_S, M_S, FK_{S1}, FK_{S2}, FK_{S3} \in S\ .\ FK_{S1}(C_S, M_S) \\ \wedge FK_{S2}(D_S, M_S) \wedge FK_{S3}(E_S, M_S)$$

In this pseudo-SQL query, A_S, B_S, C_S and M_S represents tables and $FK_{S1,}$ FK_{S2} and FK_{S3} represents foreign keys, both from the recovered schema. The result of the MDPEM over the schema represented by Fig. 10 will be composed of two occurrences. One of these results (which is only a view of the whole model) will be composed of *Classroom*, *Academic_Year, Teacher* and *Give* tables (which match the C_p, D_p E_p and M_p tables of the pattern respectively) and their corresponding foreign keys (which have no names in the schema but probably numerical identifiers). It is important to note that the result of the MDPEM can be represented by the service metamodel proposed in Fig. 7.

Implementing the patterns with an abstract description of an operation, in which all the elements of the pattern are involved, could be very useful for the semi-automatic generating of source code in further steps. In this way, after matching is done, the result is the abstract specification of an operation where the elements involved are real elements of the schema and not abstract entities of the pattern. Obviously, many parts of the abstract operation must be customized before the services are generated. This point is currently being researched and we think that would be useful to provide a special language to express these abstract operations together with our proposal.

At the end of the MDPEM process, each instance of the patterns is susceptible to being transformed to a Web Service (see the following section). Because the entire

process revolves around models and views of these models, the target implementation platform does not matter, because it is automatically generated. It only depends on the available factories to generate code in different platforms (such as J2EE, .NET, etc.).

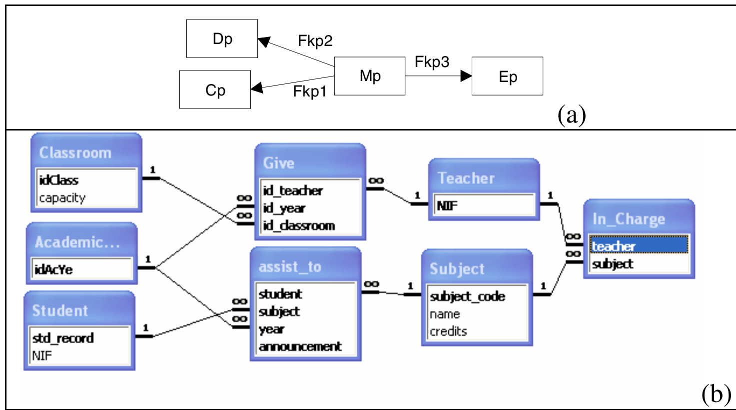

Fig. 10. Two patterns for searching in an SQL-92 Schema

7 Service Implementation: The Last Step

In the previous sections, a methodology to reengineer SQL-92 databases to discover services was explained. The main goal of this work is to develop a method to help the software engineer expose services from a legacy system, in our case an SQL-92 database.

Our choice for publishing these services is Web Services technology. The reason for using Web Services instead of other types of component or software artifacts is that Web Services were created for system integration and to wrap legacy systems [31]. Specifically, the use of Web Services for legacy system integration is a fact [32]. So, on one hand, Web Services were chosen because the target of the methodology is true legacy systems such as an SQL-92 database which is still widely in use and on the other hand, Web Services work over any technology due to the fact that this technology lean on standard protocols such as SOAP, WSDL and UDDI.

Currently, QVT transformations to generate Web Services from abstract service specifications are being defined. In this case, the core of the transformation is the obtained services and the WSDL specification of a Web Service. The WSDL documents work as a contract between the provider and the customer, because this specification is where the customer can learn which operations the Web Service provides, along with the signature of each operation. Different strategies to perform this transformation are being studied.

The source models for the last transformation in our process are the services, while the target model is the WSDL document metamodel and the implementation of the services as well. Different versions of transformations would be created in order to generate Web Services in different platforms.

8 Conclusions and Future Work

Up to now, a complex but fully functional environment for database reengineering (Relational Web) has been developed, and its efficiency has been proven in many projects [21] .

Despite the fact that SQL-92 technology is obsolete, most legacy systems and most companies are still working over this kind of database. For this reason, an effective process for reengineering this kind of database towards the web, exposing functionalities and operations, is being designed.

In section 4.1, the SQL-92 part of the metamodel presented in [25] was chosen to work with, but current results will be extended to subsequent versions of SQL (namely 1999 and 2003 versions).

Our research is mainly focused on discovering hidden functionalities by means of pattern matching and state machines. These functionalities are exposed by means of Web Services, which are also automatically generated inside the reengineering process.

Transformations are described using QVT as transformation language. Due to the novelty of this language, not too much tools support the full syntax of QVT. However, until a suitable QVT engine could be used, transformations will be represented by means of an implemented algorithm.

Acknowledgements. This work is partially supported by the MÁS project (Mantenimiento Ágil del Software), Ministerio de Ciencia y Tecnología/FEDER, TIC2003-02737-C02-02, and the ENIGMAS project, Plan Regional de Investigación Científica, Desarrollo Tecnológico e Innovación, Junta de Comunidades de Castilla La Mancha, PBI-05-058.

References

1. Blaha, M. A Retrospective on Industrial Database Reverse Engineering Projects-Part 1. in Proceedings of the 8th Working Conference on Reverse Engineering (WCRE´01). 2001. Suttgart, Germany: IEEE Computer Society.
2. ISO/IEC, ISO/IEC 9075:1992, Database Language SQL. 1992.
3. Zimmermann, O., M. Tomlinson, and S. Peuser, Perspectives on Web Services. Applying SOAP, WSDL and UDDI to Real-World Projects. Primera Edición ed. Springer Professional Computing. 2003, Germany: Springer. pp. 645.
4. Wang, X., et al. Business Rules Extraction from Large Legacy Systems. in Proceedings of the Eighth Euromicro Working Conference on Software Maintenance and Reengineering. 2004. Tampere, Finland.
5. Chikofsky, E.J. and J.H. Cross, Reverse Engineering and Desing Recovery: A Taxonomy. IEEE Software, 1990(January): p. 13-17.
6. OMG, MDA Guide Version 1.0.1. 2003, Object Management Group. p. 62.
7. Favre, J.-M., M. Godfrey, and A. Winter. Integrating Reverse Engineering and Model Driven Engineering. in Proceedings of the Second International Workshop on Meta-Models and Schemas for Reverse Engineering (ateM 2004). 2004. Delft, The Netherlands.
8. Bézivin, J. Model Engineering for Software Modernization. in Guest Talk in the 11th IEEE Working Conference of Reverse Engineering. 2004.
9. Larman, C., Applying UML and Patterns. 1998, New York: Prentice Hall, Upper Saddle River.

10. Müller, H.A., et al. Reverse Engineering: A Roadmap. in International Conference on Software Engineering - Proceedings of the Conference on The Future of Software Engineering. 2000. Limerick, Ireland.
11. Cohen, Y. and Y.A. Feldman, Automatic High-Quality Reengineering of Database Programs by Abstraction, Transformation and Reimplementation. ACM Transactions on Software Engineering and Methodology, 2003. **12**(3): p. 285-316.
12. Blaha, M. Dimensions of Database Reverse Engineering. in Fourth Working Conference on Reverse Engineering (WCRE '97). 1997. Amsterdam, The NETHERLANDS.
13. Henrard, J. and J.-L. Hainaut. Data dependency elicitation in database reverse engineering. in Fifth European Conference on Software Maintenance and Reengineering (CSMR´01). 2001. Lisbon, Portugal: IEEE Computer Society.
14. Soutou, C., Relational database reverse engineering: algorithms to extract cardinality constraints. Data & Knowledge Engineering, Elsevier Science Publishers B. V., 1998. **28**(2): p. 161-207.
15. Sousa, P.M.A., et al. Clustering Relations into Abstract ER Schemas for Database Reverse Engineering. in Proceedings of the Third European Conference on Software Maintenance and Reengineering. 1999. Amsterdam, Netherlands: IEEE Computer Society.
16. Hainaut, J.-L., et al. Database Design Recovery. in Eighth Conferences on Advance Information Systems Engineering. 1996. Berlin.
17. Henrard, J., et al. Program understanding in database reverse engineering. 2002.
18. Hick, J.-M. and J.-L. Hainaut, Strategy for Database Application Evolution: The DB-MAIN Approach. LNCS 2813, 2003: p. 291-306.
19. Thiran and J.-L. Hainaut. Wrapper Development for Legacy Data Reuse. in Eighth Working Conference on Reverse Engineering (WCRE'01). 2001. Suttgart, Alamania: IEEE Computer Society.
20. Bychkov, Y. and J.H. Jahnke. Interactive Migration of Legacy Databases to Net-Centric Technologies. in Proceedings of the Eighth Working Conference On Reverse Engineering (WCRE´01). 2001: IEEE Computer Society.
21. García-Rodríguez de Guzmán, I., M. Polo, and M. Piattini. An Integrated Environment for Reengineering. in Proceedings of the 21st International Conference on Software Maintenance (ICSM 2005). 2005. Hungary, Budapest: IEEE Computer Society.
22. Blaha, M. A Retrospective on Industrial Database Reverse Engineering Projects-Part 2. in Proceedings of the 8th Working Conference on Reverse Engineering (WCRE´01). 2001. Suttgart, Germany: IEEE Computer Society.
23. Polo, M., et al., Generating three-tier applications from relational databases: a formal and practical approach. Information & Software Technology, 2002. **44**(15): p. pp. 923-941.
24. García-Rodríguez de Guzmán, I., M. Polo, and M. Piattini. An integrated environment for reengineering. in 21st IEEE International Conference on Software Maintenance. 2005. Budapest, Hungría: IEEE Computer Society.
25. Calero, C., et al., An Ontological Approach To Describe the SQL:2003 Object-Relational Features. Accepted in "Computer Standards and Interfaces", 2005: p. 28.
26. OMG, Unified Modeling Language: Superstructure. Versión 2.0. 2005.
27. QVTP, Revised submission for MOF 2.0 Query / Views /Transformations RFP (Version 1.1). 2003, QVT-Partners (http://qvtp.org/).
28. WSSOA, Web Services and Service-Oriented Architectures. 2005.
29. OMG, Meta Object Facility (MOF) Specification. 2002.
30. Pagel, B.-U. and M. Winter. Towards Pattern-Based Tools. in Proceedings of EuropLop. 1996.
31. Alonso, G., et al., Web Services. Concepts, Architectures and Applications, ed. M.J. Carey and S. Ceri. 2004, Berlin: Springer. pp. 354.
32. Lavery, J., et al. Laying the Foundations for Web Services over Legacy Systems. in Proceedings of the Fourth International Workshop on Web Site Evolution (WSE'02). 2002. Montreal, Canada.