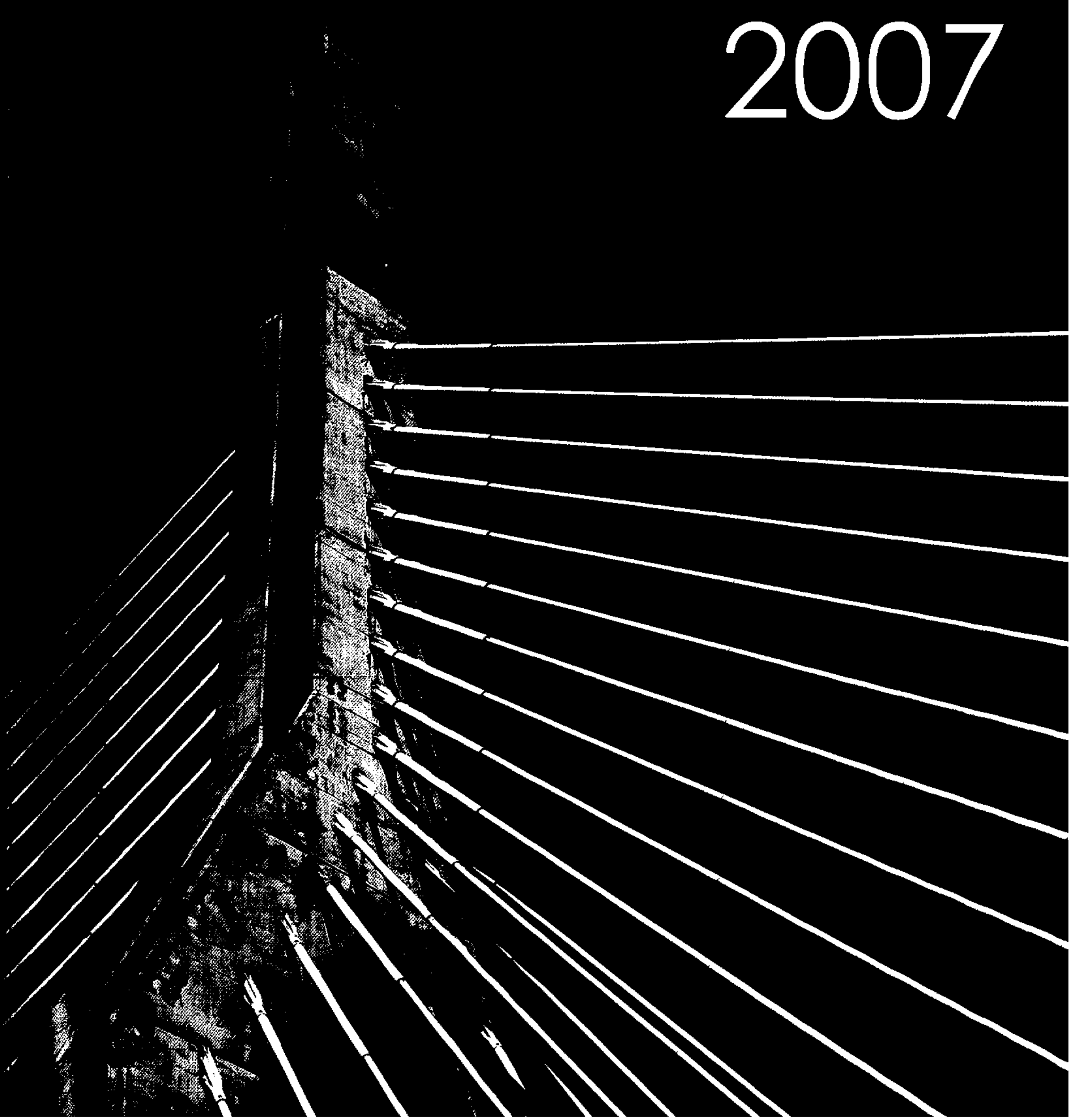
Boston, Massachusetts
July 9-11, 2007

2007



PROCEEDINGS

SEKE 2007

The 19th International Conference on Software Engineering & Knowledge Engineering

Sponsored by

Knowledge Systems Institute Graduate School, USA

Technical Program

July 9-11, 2007

Hyatt Harborside Hotel, Boston, Massachusetts, USA

Organized by

Knowledge Systems Institute Graduate School

ISBN 1-891706-20-9 (paper)

Additional Copies can be ordered from:
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076, USA
Tel:+1-847-679-3135
Fax:+1-847-679-3166
Email:office@ksi.edu
http://www.ksi.edu

Proceedings preparation, editing and printing are sponsored by
Knowledge Systems Institute Graduate School

Printed by Knowledge Systems Institute Graduate School

Table of Contents

Table of Contents

Software Processes and Engineering Practice

Aspect-Based Software Development

Software Testing and Quality Assurance

System Requirements Analysis, Modeling and Specification

Web-Based Applications

Web Technology and Web Engineering

Software Project and Resource Management

Software Reuse and Component Technology

Software System Maintenance

Knowledge Engineering, Natural Language Processing, and AI

Plenary Talk

Database Retrieval Methods

Data

Software Development and Design Pattern

Data Warehouse

Agent-Based Technology and Intelligence

DB Access and Query Processing

Service-Oriented Technology and Web Technology

System Reliability and Verification

Agent Modeling/Methodology

Agent Applications

Human Interaction and GUI Development

Service

Security

Grid Technology

Software Metrics, Measurement and Evaluation

Industrial Workshop

Note: (S) means short paper.

Using Model-Driven Pattern Matching to Derive Functionalities in Models

García-Rodríguez de Guzmán, I., Polo, M. and Piattini, M.

***Abstract*— Today, software engineering is evolving from source code to the model realm. As objects have been considered "atomic units" in software development and maintenance, models are becoming first order citizens in the well-known MDA paradigm. New languages and metamodel families are appearing to support the management of models and the new capabilities arising from this technological evolution. In this paper we present the MDPEM (Model-Driven Pattern Matching) technique and a possible application for models.**

***Index Terms*— MDA, Model instrumentation, Model-Driven Pattern Matching, OCL, QVT**

I. Introduction

Software evolution is turning classic artefacts into models as a first order element in the development process. Current development paradigms are going through a change from object orientation to model-driven trends [1]. This evolution is partially supported by many proposals such as standard languages and metamodels. Perhaps one of the most popular initiative is OMG (*Object Management Group*), consisting of UML2 (*Unified Modelling Language*) [2], OCL2 (*Object Constraint Language*) [3], MOF2 (*Meta-Object Facility*) [4], CWM (*Common Warehouse Specification*) [5] and the recently appeared KDM (*Knowledge Discovery Metamodel*) [6] (intended to support the Architecture-Driven Modernization process [7]).

Since languages such as Java, SmallTalk and C++ met the new requirements when object orientation appeared, other kinds of languages are currently appearing to fulfil the new requirements of the MDA (*Model-Driven Architecture*) [8] paradigm. These languages are capable of managing models and metamodels as other languages, such as Java or C++, deal with objects and classes.

The MDA philosophy is creating new opportunities for software engineering. For example, software development can be supported by means of models [9, 10], reengineering for software modernization [11], system integration [12], etc.

Ignacio García-Rodríguez de Guzmán is with the Alarcos Research Group. Information Systems and Technologies Department, UCLM-SOLUZIONA Research and Development Institute. University of Castilla-La Mancha, Ciudad Real, Spain. (e-mail: Ignacio.GRodriguez@uclm.es).

Macario Polo is with the Alarcos Research Group. Information Systems and Technologies Department, UCLM-SOLUZIONA Research and Development Institute. University of Castilla-La Mancha, Ciudad Real, Spain. (e-mail: Coral.Calero@uclm.es).

Mario Piattini is with the Alarcos Research Group. Information Systems and Technologies Department, UCLM-SOLUZIONA Research and Development Institute. University of Castilla-La Mancha, Ciudad Real, Spain. (e-mail: Mario.Piattini@uclm.es).

One of the most popular (and one of the most ambitious bets) to deal with models is QVT (Query/View/Transformations) [13]. This language, proposed by the OMG, is intended to perform operations with models. Using QVT, it is possible to throw queries against models, create views from models and of course, carry out transformations among models. One of the most common operations in QVT is pattern matching. For any operation, QVT expresses models as searching patterns. The QVT specification [13] establishes in which context pattern matching is used, but is does not make clear what to do when it is only necessary to look for occurrences of a given pattern.

Computer science has used pattern matching in many domains, such as lexical and syntactical analyzers, data mining, information retrieval, etc. But in most of them, pattern matching is carried out against text, files or other media. Our proposal is to use the latest advances in model technology to facilitate the application of the well-known technique of pattern matching.

This paper is organized as follows: Section II gives an overview of QVT and some related works; Section III introduces the concept of *MDPEM*; Section IV depicts what *MDPEM* could be useful for; Section V tackles a possible application of MDPEM in the aggregation of services to a system; and Section VI gives some conclusions and outlines some future lines of work.

II. Background

A. About QVT

QVT is an initiative of the OMG. In 2002, it was published the QVT RFP. One year later in March 2003, the QVT-Partners published the "Initial submission for MOF 2.0 Query/Views/Transformations RFP" [14]. In November 2003, QVT-Partners published the "Revised submission for MOF 2.0 Query / Views / Transformations RFP" [15]. This specification showed a more complete specification along with the declarative and imperative QVT's languages. In November 2005, OMG published the "MOF QVT Final Adopted Specification" [13].

In spite of the popularization of this language, there is no available any QVT engine implementing the declarative QVT language (up to now). Some projects in the academic world [16] are now underway. There are other initiatives such as the *MODELWARE QVT Tool* [17], based on the *QVT Operational* language, the other sublanguage of QVT.

B. *Pattern Matching*

Traditionally, patterns have been used for many purposes. For example, in some works [18, 19], patterns have constituted the cornerstone of the migration process from one kind of system to another. In other studies [20], the authors use a pattern-oriented language to integrate databases. Additionally, patterns have been used to reduce the inherent complexities that arise when applications must be connected to databases [20-22].

Patterns have been widely used to reengineer legacy systems [23], and particularly, patterns are very useful in the reverse engineering stage [24-26]. The well-known patterns of Gamma [27] have been used not only in the system design stage, but also with the aim of refactoring or restructuring legacy systems.

III. MODEL-DRIVEN PATTERN MATCHING

A. *What does MDPEM consist of?*

Model-Driven Pattern Matching is simply a traditional concept which has evolved together with software trends. The basic idea behind pattern matching is to find occurrences of a given pattern in a data set. Thus, MDPEM uses models to specify both the patterns and the target data set.

According to [28], *"the essential idea behind pattern matching is to allow the succinct expression of complex constraints in an input data type; data which matches the pattern is then picked out and returned to the invoker"*. In the MDPEM context, the pattern model specifies the complex constraints and the target model represents the data.

B. *A framework for MDPEM*

As noted above, this paper focuses on the *MDPEM* concept in the QVT language; thus the developed framework to perform MDPEM is based on this model-oriented language.

The QVT language uses the concept of pattern matching in every action that its constructors perform intensively. But due to the novelty of QVT, there is neither foundation nor theory about how to use this language to perform pattern matching, and, additionally, the QVT specification [13] is confusing in this respect.

For this reason, a basic framework has been developed to support (at least on a theoretical level) the MDPEM process. Fig. 1 depicts the framework. For the sake of simplicity all the involved artifacts are located in an MOF-like pyramid. Level M2 of Fig. 1 represents the metamodels for patterns and target models, as well as the model transformations involved in the process; Level M1 represents all the models involved in the MDPEM process, namely the patterns, the target model, and the matchings (occurrences of the pattern in the target model). Since all the steps are developed in M2 and M1, M3 and M0 are not explained. Table I summarizes the elements represented in Fig. 1. For each element (*"Level M2"* column in TABLE I) on the M2 level, there exists a conforming model (*"Level M1"* column in TABLE I) on the M1 level.

The *PatternToQVTPattern* transformation, not included in Table I, plays an important role in the MDPEM process. As noted above, any pattern (*PatternModel*) must be transformed into a different representation that QVT imposes: the *QVT Template* (shown as *QVTPattern Model* in Fig. 1). The *QVT Template* is the structure that QVT uses to represent any pattern that needs to be matched. As a consequence, any pattern we want to match should be transformed to this representation. Therefore, a transformation (*PatternToQVTPattern* in Fig. 1, *Level M2*) involves both the pattern metamodel (PatternMModel) and the QVTPattern MModel (QVT Template Metamodel).

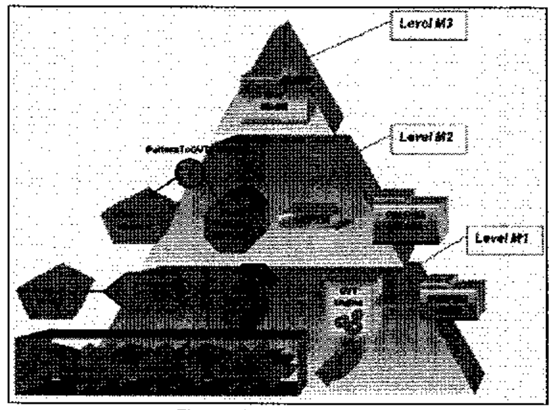


Fig. 1. MDPEM Framework

TABLE I
ELEMENTS INVOLVED IN THE MDPEM PROCESS

Level M2	Level M1	Description
PMM	PM	Patterns (models) are constrained by a given metamodel. This must be compatible with the target model to be able to find matchings.
QVTPMM	QVTPM	Patterns must be translated into a QVT pattern representation (QVT Template [13]), so the QVT Template metamodel (QVTPMM) should be taken into account. As a consequence, each pattern model is transformed to a QVT Template (QVTPM)
PIM/PSMMM	PIM/PSMM	Because in any MDA process models are called PIM or PSM (according to the abstraction level of the model), we suppose that the target model will be a PIM or a PSM
-	Matching	After MDPEM is applied, matchings may be found. This matchings can be seen as fragments of the target model, so the metamodel for these elements can also be the PIM/PSMMM

PMM = Pattern Metamodel; PM = Pattern Model; QVTPMM = QVT Pattern Metamodel; QVTPM = QVT Pattern Model; PIM/PSMMM = PIM/PSM Metamodel; PIM/PSMM = PIM/PSM Model

The MDPEM process can be divided into four steps:

1. Pattern model and target model are defined.
2. QVT Template model is obtained from Pattern Model.
3. MDPEM is carried out.
4. Matchings (sub-models) are returned to the invoker.

For a deeper explanation of MDPEM and examples of patterns, please see [29].

IV. Finding/Defining Defining Pattern on Models

The previous section outlined the MDPEM technique, but the main goal of this technique is not only to find occurrences for a given pattern. MDPEM can be also used to undertake more detailed operations over those models acting as target models.

This technique was conceived as a tool to support some parts of an MDA process to extract services from relational databases [30]. In this process, patterns are used to extract functionality from relational databases. That led us to the question: how to use MDPEM to do something more than find matchings?

A. *Patterns plus actions to derive functionalities*

In this paper our aim is to provide a mechanism to use MDPEM undertaking actions associated with the set of occurrences obtained in the matching process (Fig. 2).

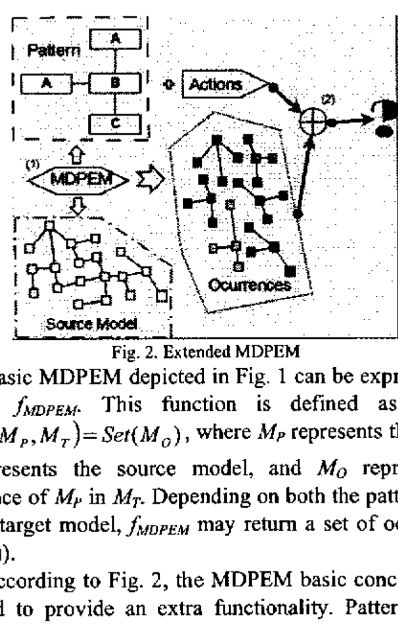


Fig. 2. Extended MDPEM

The basic MDPEM depicted in Fig. 1 can be expressed as a function f_{MDPEM}. This function is defined as follows: $f_{MDPEM}(M_P, M_T) = Set(M_O)$, where M_P represents the pattern, M_T represents the source model, and M_O represents an occurrence of M_P in M_T. Depending on both the pattern model and the target model, f_{MDPEM} may return a set of occurrences ($Set(M_O)$).

But according to Fig. 2, the MDPEM basic concept can be extended to provide an extra functionality. Patterns can be provided together with a description of an abstract action. The basic idea behind Fig. 2 is to have not only a pattern (M_P) but also an associated abstract action, which could be later transformed into a concrete transformation.

However, MDPEM can be specified in a more powerful way. First of all, instead of a pattern M_P, the process takes a pair <*pattern, abstract action*>, algebraically represented as $K = (M_T, A_A)$, where M_T represents the aforementioned pattern model, and A_A represents the abstract action. Therefore, the previously defined function f_{MDPEM} must be redefined according to the new element K. This new function can be considered as follows: $f_{MDPEM}(K, M_T) = Set(A_A(M_O))$. A_A is then applied to each element in the resulting set of occurrences. Thus the abstract action, which is defined in terms of the abstract elements of the pattern, is then "materialized" for each actual occurrence obtained in the MDPEM process.

In Fig. 2, K is represented by the pattern plus the action (top of the figure), and M_T is the source model depicted at the bottom of figure. After applying MDPEM and obtaining the occurrences, the specified action is then applied to each occurrence.

The term "abstract action" may be very vague, but in fact any action or structure can be specified to be applied to each occurrence. As a consequence, certain parts from models (MDPEM occurrences) can be updated or modified according to the instructions given in the abstract action (A_A).

In the following section, we will describe what MDPEM has been used for: to detect services in databases using patterns and abstract descriptions for the services attached to patterns.

V. Using MDPEM to Find and Create Services

This section explains the purpose of this technique. As noted, MDPEM was developed in the context of an MDA process to extract services from relational databases. The lack of tools and experience around languages such as QVT prompted the development of MDPEM to properly manage models and patterns.

In the context of the process proposed in [30], we can define a pattern and attach a particular behavior to this pattern. This behavior is specified in terms of operations, called services. Thus, for a given pattern involving a set of entities, a set of abstract operations involving these entities can be attached. Thus, when an occurrence is found, each abstract operation attached to the pattern is "instantiated" with the entities of the occurrence.

A. *Problem definition*

MDPEM can be used to discover services in relational databases. Thus, all the artifacts (patterns and target model) will be described in terms of tables, columns, foreign keys and so on.

The aim is to start from a database model, define a set of patterns, find their matches and obtain possible useful services. This case study fits the example depicted in Fig. 2, and:

- the database model plays the role of the source model,
- a pattern describing a particular database fragment structure plays the role of the pattern, and
- the abstract description of an operation involving the elements of the pattern.

B. *Pattern model and target model definition*

Fig. 3 depicts a fragment from a relational database extracted from an industrial application. This fragment is composed of 10 tables and will form the *target model* for the MDPEM process. This model has been used to carry out a simple case study and illustrates the work presented in this paper. Due to the lack of space, the picture does not include any detail such as columns, primary keys or the like. It can be assumed that a suitable database metamodel is used (in fact, a summarized version of the one presented in [31] is used).

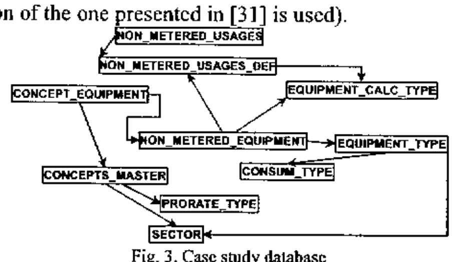


Fig. 3. Case study database

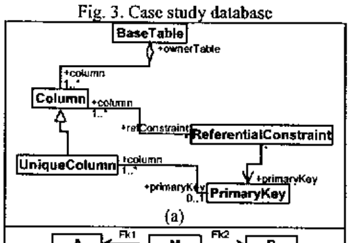


Fig. 4. (a) Pattern metamodel, (b) DFK (Double Foreign Key) Pattern, and (c) DFK pattern conforming to pattern metamodel

Fig. 4 (a) represents the pattern metamodel. Since it may not be sufficiently descriptive, Fig. 4 (b) shows a conceptual representation. Fig. 4 (b) shows the pattern to be applied in the MDPEM process.

Once the source model and pattern are defined, the abstract operation must be defined (the "Action" in Fig. 2). In this context, any service (or operation) against the database is translated into an SQL query. But in spite of existing proposals to represent SQL queries in a metamodel-like representation (PL/SQL Grammar[1]), there is no standardized DML metamodel for SQL. Nonetheless, a suitable solution lies in the OMG's metamodel family, OCL2. OCL2 is as expressive as SQL. Furthermore, the OCL2 specification includes the language metamodel, so that any OCL expression can be represented as a model [32] and thus used in any MDA approach as an artifact. For these reasons, OCL was chosen to describe the abstract operations.

In the pattern (Fig. 4 (b)), table M has two foreign keys: one to table *A* and one to table *B*. The *M* primary key is obtained by linking together the foreign key for table *A* and the foreign key to table *B*. Thus, for a given value of the *B* primary key it is possible to obtain those records from table *A* linked to table *B*. As a consequence, two operations or services can be attached to the pattern in Fig. 4 (b):

- *getA_TupleBy_B(b:B):bag(A)* → Having the values of the *B* table tuple, it is possible to obtain a list of tuples from A by means of table *M*.
- *getB_TupleBy_B(a:A):bag(B)* → As in the previous operation, here it is possible to return a set of tuples to table *B*. It is only necessary to pass the value of an *A* table primary key to perform the search.

Fig. 6. (a) includes the required OCL postcondition to specify the service *getA_TupleBy_B(b:B):bag(A)*. Supposing that these services are allocated in a façade class [27], the OCL expression:

1. Recovers all the instances (M is a class representing a table) of *mM* property,
2. for each instance:
    a. calculate the values or properties which implement the foreign key to *B*,
    b. calculate the values of the properties conforming the primary key of *b* (argument of the operation),
    c. calculate which of these sets have the same values, and
3. for each of these sets, return the corresponding *mA* attribute in the *M* context.

The Fig. 6 (b) represents the same expression, but in a model-like representation. It should be noted that all the elements referred in the expression belong to the pattern (see Fig. 4 (b)).

C. *QVT Template generation*

Once all the elements involved in the process have been defined, the pattern must be transformed into a suitable representation to perform the MDPEM process.

```
Pattern =
{
    e1: <classname1>; e2: <classname2> .... eN:<classnameN>
    l1 : <assoc1> (ei, ej) .... lm:<assocM>(eu, ew)
    where <predicate>
}
```

Fig. 5. *QVT Template* abstract representation

As noted above, QVT defines its own pattern representation: the *QVT Template* [13] (also called *QVT Pattern Model* in Fig. 1). A *QVT Template* (see Fig. 5) represents a pattern on the basis of the elements involved in the pattern and the relationships between them. Thus, before performing the MDPEM process the input pattern must be transformed into a *QVT Template*. In the proposed framework (see Fig. 1), this transformation is carried out by means of a QVT transformation (*PatternToQVTTemplate*).

This transformation, outlined in Fig. 7, transforms patterns (based on the metamodel in Fig. 4 (a)) to a *QVT Template* representation. Thus, starting from a pattern and then applying this transformation, a suitable QVT Template is obtained to perform MDPEM. The Fig. 7 transformation is divided into five relations. Each relation is intended to address the transformation from each part of the original metamodel (Fig. 4 (a)) to other part in the QVT Template one (Fig. 5). See [29] for a further description of this transformation.

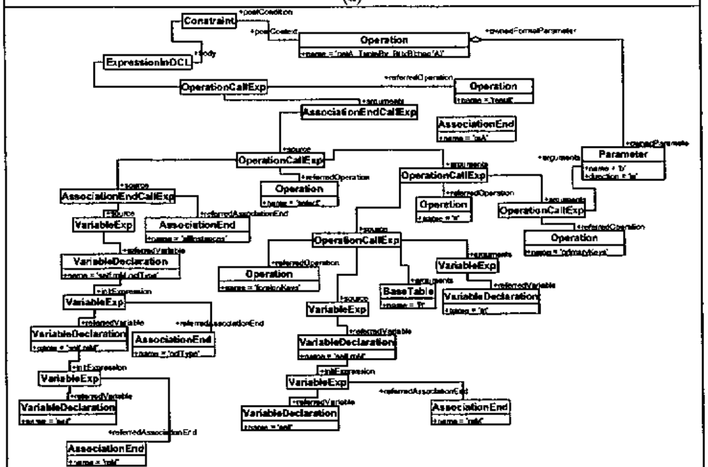
D. *MDPEM application and matchings generation*

This step depends entirely on the QVT engine that executes the matching. As can be seen in Fig. 1, the QVT engine takes the source model (referred in the picture as the "PIM/PSM Model") and the QVT Template (obtained from the initial pattern) as input. Then, the QVT engine executes the matching and automatically returns the occurrences.

[1] https://javacc.dev.java.net/files/documents/17/2959/FormsPlSql.jj

```
result = self.mM.oclType.allInstances
     ->select(m:M|foreignKeys(m,mB) = b.primaryKeys()).mA
```

(a)



(b)

Fig. 6. (a) OCL postcondition for the *getA_By_B(p:B):Set(A)* and (b) its model-based representation

The obtained matchings can be seen as constrained views of the target model. These small "sub-models" represent something interesting from point of view of the engineering which is supervising the MDPEM process. In this work, we present one of the possible uses of MDPEM: the inclusion of operations in the recovered matchings.

```
transformation SQL92SchemaPatternToQVTTemplate
   (sql92p: SQL92SchemaPattern, qvtt:QVTTemplate)
{
key ObjectTemplateExp = (name);
key PropertyTemplateItem = (name);
top relation BaseTableToObjectTemplateExp{...}
top relation ColumnToObjectTemplateExp{...}
top relation UniqueColumnToObjectTemplateExp(...)
top relation ReferentialConstraintToObjectTemplateExp{...}
top relation PrimaryKeyToObjectTemplateExp{...}
}
```

Fig. 7. QVT transformation to generate a QVT Template from a patter for database models

As depicted in Fig. 2, patterns are given together with the abstract description of an operation (shown, for example in Fig. 6). After MDPEM is applied and the matchings are recovered, the abstract operation is applied to each obtained occurrence. Thus, since the elements of the matchings belong to the target model, the elements involved the abstract operation will no longer be abstract.

TABLE II
MATCHINGS OBTAINED IN THE MDPEM PROCESS

ID	A	M	B
1	CONCEPTS_ MASTER	CONCEPT_ EQUIPMENT	NON_METERED_ EQUIPMENT
2	CONSUM_TYPE	EQUIPMENT_TYPE	SECTOR
3	PRORATE_TYPE	CONCEPTS_MASTER	SECTOR

For the given pattern in Fig. 4 (a), and the target model in Fig. 3, the set of matchings after applying MDEM are described in the TABLE II, and the resulting services in TABLE III.

TABLE III
SERVICES OBTAINED FROM MATCHINGS AND ABSTRACT OPERATIONS

ID	SERVICE
1	-*getCONCEPTS_MASTER_TuplesBy_NON_METERED_EQUIPMENT* - *getNON_METERED_EQUIPMENT_TuplesBy_CONCEPTS_MASTER*
2	- *get*CONSUM_TYPE_TuplesBy_SECTOR - getSECTOR_TuplesBy_CONSUM_TYPE
3	- *getPRORATE_TYPE_TuplesBy_SECTOR* - *getSECTOR_TuplesBy_PRORATE_TYPE*

Of course, the names of the services in TABLE III are not intuitive and the names of the abstract operations are just illustrative. Services created from matchings and abstract operations must be renamed by the engineer, according to the service's functionality. Therefore, services become intuitive and easy to use. For each service in TABLE III, references to abstract elements (namely A, B, and M) are substituted by the elements in the matching playing this "abstract role" (e.g., in TABLE II, in the occurrence with ID = 2, "*SECTOR*" plays the role of *B*, "*CONSUM_TYPE*" the role of *A*, and "*EQUIPMENT_TYPE*" the role of *M*.

Due to the lack of space, an abstract operation "instantiated" for a matching could not be shown.

VI. CONCLUSIONS AND FUTURE WORK

This paper presents MDPEM, an approach to perform pattern matchings in the context of models. The proposed framework focuses on QVT, a particular language to manipulate models by means of queries, views and transformations.

In addition to MDPEM, this paper focuses on what MDPEM is able to do with models. The main idea behind MDPEM is to use the models obtained as a result of the matchings to perform an action over these models.

Our aim is to use the dual "pattern-abstract operation" against a target model. Thus, portions of the target model (the occurrences of the pattern) can be annotated with operations or services (described as abstract operations related to the searching pattern). The instrumentation of models with these services facilitates the implementation of MDA-based development processes.

As well as development, MDPEM might be useful in other areas. Software maintenance in a model-level can be improved using this technique, e.g., looking for design patterns, as well as looking for bad-smells and correcting them (in a model – level). Every process described as a "*search pattern and apply rule*" sequence can be partially automated and then easily applied.Due to the lack of tools for implementing the QVT language, MDPEM has not been tested in an automated environment. As soon as a tool is available, this theoretical proposal will be migrated to a *runnable* one

For each pattern metamodel, it is necessary to develop a suitable transformation to generate the equivalent *QVT Template* model. Therefore, it may be useful to automate the generation of the *QVT Template* from some pattern metamodels.

ACKNOWLEDGMENTS

This work was partially supported by the ENIGMAS poject (PIB-05-058), the FAMOSO Project (2006: FIT-340000-2006-67); and the MECENAS project (PBI06-0024).

REFERENCES

[1]. Bézivin, J. *Introduction to Model Engineering*. 2006. <www.modelware-ist.org> [Accesed: 14-November-2006]

[2]. OMG. *Unified Modeling Language: Superstructure. Versión 2.0*. 2005. <http://www.omg.org/docs/formal/05-07-04.pdf> [Consultado el: 21 de Noviembre del 2005]

[3]. OMG, *OCL 2.0 Specification. Version 2.0*. 2005, Object Management Group (OMG). p. 185.

[4]. OMG. *Meta Object Facility (MOF) Specification*. 2002. <http://www.omg.org/docs/formal/02-04-03.pdf> [Accessed: 11-05-2005]

[5]. OMG. *Common Warehouse Metamodel (CWM) Specification*. 2003. <http://www.omg.org/docs/formal/03-03-02.pdf> [Consultado el: 29-09-2005]

[6]. OMG. *Architecture-driven Modernization (ADM): Knowledge Discovery Metamodel (KDM) Specification*. 2006. <http://www.omg.org/docs/ptc/06-06-07.pdf> [Accessed in February, 2006]

[7]. OMG. *ADM Task Force*. 2006. <http://adm.omg.org> [Accessed in February 2007]

[8]. OMG, *MDA Guide Version 1.0.1*. 2003, Object Management Group. p. 62.

[9]. Maamar, Z., et al. *Towards a contextual model-driven development approach for Web services*. in *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006)*. 2006. Pahos, Cyprus

[10]. Xiaofeng, Y., et al. *A Model Driven Development Framework for Enterprise Web Services*. in *Proceedings of the 10th Conference on Enterprise Distributed Object Computing (EDOC'06)*. 2006. Hong Kong, China: IEEE Computer Society IEEE Computer Society. ISBN. 0-7695-2558-X

[11]. Reus, T., H. Geers, and A. van Deursen. *Harvesting Software for MDA-Based Recovering*. in *European Conference on Model Driven Architecture - Foundations and Applications*. 2006. Bilbao (Spain): Springer-Verlag Berlin Heidelberg. **LNCS 4066** Springer-Verlag Berlin Heidelberg, A. Rensink and J. Warmer, Editors. ISBN. 3-540-35909-5

[12]. van den Heuvel, W.-J. *Matching and Adaptation: Core Techniques for MDA-(ADM)-driven Integration of new Business Applications with Wrapped Legacy Systems*. in *Model-Driven Evolution of Legacy Systems (MELS 2004)*. 2004. Monterey, California, USA

[13]. OMG. *MOF QVT Final Adopted Specification*. 2005. <http://www.omg.org/docs/ptc/05-11-01.pdf> [Accessed in February, 2005]

[14]. OMG. *QVT-Partners initial submission to qvt-rfp*. 2003. <http://www.qvtp.org/downloads/1.0/qvtpartners1.0.pdf> [Accessed in November, 2006]

[15]. OMG. *Revised submission for MOF 2.0 Query/Views/Transformations RFP*. 2003. <http://www.omg.org/docs/ad/04-04-01.pdf> [Accessed in November, 2006]

[16]. Queralt, P., et al. *Un Motor de Transformación de Modelos con soporte para el Lenguaje QVT Relations*. in *Desarrollo del Software Dirigida por Modelos y Aplicaciones 2006 (DSDM'06)*. 2006. Sitges, Barcelona (Spain)

[17]. ModelWare. *The ModelWare QVT Tool*. 2007. <http://www.modelware-ist.org/> [Accesed: 15-02-2007]

[18]. Bodhuin, T., E. Guardabascio, and M. Tortorella. *Migrating COBOL Systems to the WEB by Using the MVC Design Pattern*. in *Ninth Working Conference on Reverse Engineering (WCRE'02)*. 2002. Richmond, Virginia: IEEE Computer Society IEEE Computer Society

[19]. Goedicke, M. and U. Zdun, *Piecemeal legacy migrating with an architectural pattern language: a case study*. Journal of Software Maintenance and Evolution: Research and Practice, 2002. **14**: p. 1-30.

[20]. Brown, K. and B.G. Whitenack, *Crossing Chasms, A Pattern Language for Object-RDBMS Integration*, ed. K.S.C. Editor. 1995. K.S.C. Editor.

[21]. Yoder, J. *Patterns for making business objects persistent in a relational database*. in *Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 2002. Tampa Bay (Florida)

[22]. Yoder, J., R.E. Johnson, and Q.D. Wilson. *Connecting Business Objects to Relational Databases*. in *Proceedings of the 5th Conference on the Pattern Languages of Programs*. 1998. Minticello-IL (Estados Unidos)

[23]. Cagnin, M.I., et al. *Reengineering Using Design Patterns*. in *Seventh Working Conference on Reverse Engineering (WCRE'00)*. 2000. Brisbane, Australia: IEEE IEEE

[24]. Sartipi, K. and K. Kontogiannis. *On Modeling Software Architecture Recovery as Graph Matching*. in *International Conference on Software Maintenance (ICSM'03)*. 2003. Amsterdam, The Netherlands: IEEE Computer Society IEEE Computer Society

[25]. Pinzger, M., et al. *Revealer: A Lexical Pattern Matcher for Architecture Recovery*. in *Proceedings of the 9th Working Conference on Reverse Engineering*. 2002. Richmond, Virginia: IEEE Computer Society IEEE Computer Society

[26]. Stoermer, C., L. O'Brien, and C. Verhoef. *Practice Patterns for Architecture Reconstruction*. in *Proceedings of the 9th Working Conference on Reverse Engineering (WCRE'02)*. 2002. Richmond, Virginia: IEEE Computer Society IEEE Computer Society

[27]. Gamma, E., et al., *Design Patterns Elements of Reusable of Object-Oriented Software*. 1995: Addisson-Wesley.

[28]. QVTP. *Revised submission for MOF 2.0 Query / Views /Transformations RFP (Version 1.1)*. 2003. <http://tratt.net/laurie/research/publications/papers/qvtpartners1.1.pdf> [Consultado el:23-06-2005]

[29]. García-Rodríguez de Guzmán, I., M. Polo, and M. Piattini. *A Framework for Model-Driven Pattern Matching*. in *Proceedings of the 9th International Conference on Enterprise Information Systems*. 2007. Funchal, Madeira - Portugal (In press)

[30]. García-Rodríguez de Guzmán, I., M. Polo, and M. Piattini. *A Methodology for Database Reengineering to Web Services*. in *European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2006)*. 2006. Bilbao (Spain): Springer-Verlag Berlin Heidelberg. **LNCS 4066** Springer-Verlag Berlin Heidelberg, A. Rensink and J. Warmer, Editors. ISBN. ISBN 3-540-35909-5

[31]. Calero, C., et al., *An Ontological Approach To Describe the SQL:2003 Object-Relational Features*. Accepted in "Computer Standards and Interfaces", 2005: p. 28.

[32]. Reynoso, L., M. Genero, and M. Piattini. *OCL2: Using OCL in the Formal Definition of OCL Expression Measures*. in *Quality in Modeling (QIM'2006)*. 2006. Genova (Italy)