From Schema and Model Translation to a Model Management System

Paolo Atzeni, Luigi Bellomarini, Francesca Bugiotti, and Giorgio Gianforme

Dipartimento di informatica e automazione
Università Roma Tre
{atzeni}@dia.uniroma3.it,
{bellomarini,bugiotti}@yahoo.it,
{giorgio.gianforme}@gmail.com

Abstract. Model management addresses problems dealing with forms of collaboration among heterogeneous databases. This collaboration may include exchange of data, schema integration, synchronization, translation and, in general, any issue characterized by a data evolving scenario. It provides a structured framework allowing standard solutions to data programmability problems in terms of the application of some recurring operators. The main mid-term target in this field is the definition of a model management system, a software platform providing the data architect with a complete set of tools addressing a wide spectrum of possible problems. In this paper we recall MIDST, a platform that works as an applicator of schema transformations. It was firstly conceived to perform model-independent schema and data translation. Then it has been extended to an applicator of general schema transformations including model management operators. Leveraging on MIDST rich representation of models, schemas and data based on a metalevel approach, we reason about potentialities and possible developments of this platform with the target of laying the basis for a real runtime model management system.

Keywords: model management, model management system, model-independent schema and data translation, data programmability.

1 Introduction

The management of heterogeneous databases, in integrated or collaborative contexts, always involves the need for solutions to *data programmability* issues. In general, data programmability addresses problems dealing with evolving scenarios: changes in a database which collaborates in a heterogeneous environment often implies a sequence of propagating changes in related databases at any level, model, schema and data [8,13,14]. Heterogeneity means that on the one hand systems are developed by different people, fostering different data models and technologies; on the other hand it recalls problems involving different software components using shared and interoperating data.

Moreover, business requirements which guided the design of the data model are not static but change in time, leading to an intensive refactoring, redefinition and migration of data. This even complicates data programmability, opening to a range of further problems including change propagation, synchronization, data exchange, integrity constraint satisfaction, data provenance memorizing and so on [7,6].

Model management provides a structured framework to encompass all these problems and establishes standard solutions based on the application of a finite set of operators [7].

There is the tendency to recognize the need for a model-independent solution to model management problems. Our approach had been pursuing model-independent schema and data translation [3,5,22,24] and led to the platform MIDST [3,4]. It was originally conceived only as an implementation of the MODELGEN operator (the one responsible for schema translation), but to a wider extent MIDST can be now considered as a general purpose applicator of schema transformations. By proposing a model-independent but model-aware definition and implementation of most common model management operators, we are working on a blueprint for standard solutions to recurring problems such as round-trip engineering and forward engineering [1]. These solutions refer to an off-line approach and not to a real runtime environment in which they are actually needed.

In the perspective of turning MIDST into a real model management system (MMS) [8], providing the data architect with a complete suite of tools to cope with data programmability problems in a real runtime context, here we address the main challenges in this migration.

We focus on the major scenarios where MIDST may be applied, as well as on the complementary issues that such a change of perspective raises. We underline how MIDST metalevel representation of models and schemas supports meaningful potentialities. We show how the main challenges can be either directly addressed by the platform, or faced through appropriate extensions that benefit from the metalevel expressivity.

We discuss the application of MIDST to the handling of evolving scenarios. Hence we explain possible approaches towards problems such as update propagation, schema synchronization and data exchange: reasonings about how model-independent solutions to simpler problems can be combined and successfully applied to these situations are provided. Complementary problems and refinements providing the data with higher quality are described as well. Special attention is devoted to data provenance treatment and integrity constraint handling, where MIDST metalevel expressivity is particularly effective.

Reasonings dealing with a concrete and advanced application of the framework to an object-oriented scenario is provided. We introduce a possible development of the platform, enabling a transparent handling of object-oriented structures.

The remainder of the paper is organized as follows. In Section 2 we describe the approach to data model handling used by MIDST; in Section 3 we outline the fundamental challenges in turning MIDST into a real MMS; in Section 4 and 5 we introduce some data programmability issues that can be addressed exploiting

MIDST metalevel potentialities; Section 6 illustrates an object-oriented scenario; finally Section 7 concludes the paper.

2 Background

In previous papers [3,4] we proposed MIDST, a platform for model-independent schema and data translation. The framework is based on the fundamental observation that any existing data model can be represented with a finite set of constructs [17]. Therefore MIDST handles a *metamodel* allowing the definition of general purpose constructs called *metaconstructs*. They are then used to assemble models, meaning that a model is thus defined as a collection (a subset) of all the existing metaconstructs. Schemas are consequently defined with respect to the model they belong to and have concrete constructs which inherit their properties from the metalevel.

Metaconstructs are characterized by a unique identifier (OID), a defining name, a set of properties coding details of interest and a set of references relating them to one another. Graphs of interrelated constructs build a model. For example we have a construct named abstract which models any "abstract" conceptual entity, such as objects (of the object-oriented model), entities (of the ER model), and so on. MIDST also manages aggregations which are constructs representing table-like entities (like tables in the relational model). Instead a lexical is a metaconstruct representing a lexical value independently of the model of interest. Then, whereas an abstract is simply characterized by its name, a lexical also has some defining properties such as whether it is identifier or not, whether it is nullable or not and so forth. We emphasise that many constructs are related to other constructs: an example is the lexical which can be linked either to an abstract (coding for example an attribute of entity) or to an aggregation (representing a relational column) by means of typed references.

The metalevel is implemented by means of a *multilevel relational dictionary* [2] which models all the mentioned concepts as relational tables: metaconstructs, properties and references.

Another important concept in MIDST is that of *supermodel*: it represents the most general model, including all the possible metaconstructs. Hence any other model is a specialization of it. The supermodel is the component that actually allows the model-independent schema translation that can be formulated in three phases: the schema (instance of the source model) is copied into the supermodel; a translation into the target model takes place in this environment; the result schema is finally downcast into the destination model. Translations are specified by means of datalog rules defined over MIDST metaconstructs.

3 Towards an MMS

3.1 Model-Independent Operators

MIDST was conceived in order to provide an implementation of MODELGEN, the operator responsible for schema translation. More formally, given a schema S_1

of a model M_1 and a target model M_2, MODELGEN computes a schema S_2 of model M_2 which corresponds to S_1. The translation process may be divided into two phases: the first one where the appropriate datalog translation rule is chosen and the second one, where the manipulations are actually performed.

We are currently working on the extension of this platform [1] to implement most important model management operators. In fact we can consider a datalog translation rule as a general transformation that can be applied to schemas. As a consequence, MIDST can be thought of as a platform capable of performing transformations that are not necessarily translations, but general model management operators.

These operators are coded with datalog rules, expressing manipulations directly in terms of the metaconstructs. This implies that they do not depend on the specific model of application. In particular, these datalog rules can be written to implement the DIFF and the MERGE operator: the first one computes the difference between two given schemas, while the second one performs a set-oriented union.

The general datalog implementation of the operators is composed of two phases. The first one works out the correspondences among the constructs of the source schemas. Then the second performs the specific operations. The DIFF, given two input schemas, copies into the target all the constructs belonging to the first, but not to the second one on the basis of the computed correspondences. Similarly, the MERGE copies into the target schema the constructs of both the source schemas; it uses the correspondences in order to avoid duplicates in the result.

Moreover the operators can be automatically generated from the currently available supermodel. We have a procedure that detects all the possible constructs and their relationships by interacting with the metalevel, and thus generates globally valid operators.

Solutions to most common model management problems can be expressed as scripts composed of a sequence of operators. Indeed, MIDST can be currently used to provide an off-line solution to some important problems such as *forward engineering* and *round-trip engineering*. As for the former, given a specification schema S_1 and an implementation schema I_1 which derives from S_1, changes in S_1 leading to a modified specification S_2 have to propagate to I_1 in order to obtain a coherently modified implementation I_2. Instead, as for the round-trip engineering: given a specification schema S_1 and an implementation schema I_1 which derives from S_1, changes in I_1 leading to a modified implementation I_2 have to propagate backwards to S_1 in order to obtain a coherently modified specification S_2.

Moving from Bernstein's solving procedures [7], we can build scripts for the cited problems in terms of the DIFF, MERGE, MODELGEN and in some sense MATCH operators.

3.2 Larger Scale and Complementary Issues

Solutions to model management problems are built in terms of scripts involving the application of several operators. Indeed, this is still a simplified version of what is actually needed in a concrete application context. In that wider perspective, issues are composed of sets of elementary model management problems whose solutions have to be coordinated.

We may consider several levels of abstraction: model management operators (such as DIFF and MERGE) belong to the lowest one, then we have simple problems (such as forward and round-trip engineering) with solutions in terms of simple operators; the top level contains more complex scenarios, such as ETL processes, whose solutions can be obtained only by coordinating procedures addressing simpler problems. Therefore, the ability to solve model management problems by building scripts in terms of simple operators is only the first requirement of an MMS, it is only the implementation of the lowest abstraction level. A meaningful MMS must cope with every level and then allow for high-abstraction environments that coordinate several model management problems.

This might seem only a change of scale issue, however it also raises collateral requirements and complementary problems that must be separately faced. Some examples of these issues include: the need for information about the provenance of data; a coherent handling of integrity constraints; a fine-grained access control; an efficient indexing mechanism for the data sources; triggers and business logic integration.

Besides, there are several quality requirements a real MMS should also consider: first of all runtime support but also availability, supportability, performance, security. As far as quality requirements are concerned, we can assume that in a research project they do not represent the riskiest element, in fact a good architectural definition phase can cope with them, leading to the design of the appropriate components.

However a simple off-line characterization of model management problems, although within a well-designed architecture, is not functionally sufficient. Indeed, MIDST currently implements the lowest abstraction layer, so it represents the first phase of the development of a real MMS.

Our aim is to discuss the most meaningful issues in the process of turning MIDST into a full-featured MMS. Hence in the following section we deal with some larger-scale problems to exemplify our approach; moreover we give details about possible directions to address complementary requirements.

4 Handling Evolving Scenarios

Evolving scenario problems [8] subsume issues dealing with changes performed over some given schemas or instances. In general, changes imply a complex net of propagations updating a set of interrelated databases.

Update propagation problems can be considered as a generalized and runtime extension of the forward engineering, involving that a change on a given database leads to modifications in another one. From a technological point of view, whereas currently MIDST solves the round-trip and forward engineering with respect to imported schemas (and hence off-line), this problem needs an on-line binding of two schemas so as to propagate changes during the execution.

A more complete definition of such a problem is *schema synchronization* which actually involves a propagation of the changes between two schemas in both directions. It may be worked out by MIDST either with a reversed application of the forward engineering solving script, or through the application of it in the one direction and by the use of round-trip in the other.

Peer to peer mappings define a net of databases whose schemas are interrelated. Therefore changes on one schema induce a chain of possibly different modifications in other schemas. This problem can be dealt with by MIDST as well: the core point is the definition of relations between databases (mappings) that can be directly applied to target schemas; it is important to point out that MIDST already solves something similar when performs a chain of schema translations. Beacause of the general treatment of transformations, such an approach may be suitable in this context as well.

Data exchange problem is aimed at transforming a mapping between two schemas into a directly executable query actually moving data. With respect to this problem, MIDST paradigm supports advantageous preconditions [8]: only relational schemas are treated and the mappings are conjunctive queries. In fact here a relational meta-representation of any model is possible due to the dictionary generality, besides we use datalog mappings that are declarative rules in the form of conjunctive queries.

However, some remarkable issues in facing these problems are still open. In the general solution to round-trip and forward engineering provided by MIDST, we assume a substantial coincidence between schema and data evolution. It means that modifications between two different databases are mainly schema-driven. The migration of instances is handled as a consequence of the evolution of schemas. Datalog rules defining manipulations on the schemas are syntactically translated into transformations on the corresponding instances.

It implies that if a logical coherence between them exists, then operators correctly perform; otherwise the tight coupling between the two levels could become a drawback. In addition there are problems, such as data exchange, that are specifically instance-oriented. In both the cases we need to loosen that coupling.

The schema-instance coupling problem is closely related to representing mappings. Many model management operators need information about schema correspondences in order to operate on them. For instance a difference operator must know those matches in order to correctly subtract constructs. In our platform, we do not currently handle an explicit representation of mappings. We move from a *unique name assumption* stating that metaconstructs with equal lexical properties are equal themselves. Under this assumption, model management operators perform a hierarchical comparison between input schemas in order to correctly treat them.

Again, this outlines the fact that our operators are schema oriented. Certainly it is a remarkable strength of the approach, providing generality thanks to the model-independent approach. Yet when data-oriented problems are faced or the treatment desired for instances differs from the one wanted for schemas, the need for some kind of decoupling arises.

One solution may involve the adoption of two different transformation languages: one for schemas and one for instances. It would involve a complete decoupling between the two layers, forcing the platform to separately deal with two levels of abstraction and, as a consequence, with two types of transformation rules. The opposite strategy is the one currently adopted by MIDST (with the cited drawbacks): one unified language for both schema and instances that, actually, causes schema rules to be initially translated into instance rules.

The two extreme views can be reconciled in two ways: designing an intermediate language allowing for greater expressivity on instances or modelling an engineered representation for mappings.

The first branch of solutions can be again split in two different approaches. One could think of an extension to schema transformations rules involving a set of functions specifying expressive manipulations for instances. In this way a more straightforward generation of data rules from schema rules would be allowed: it would not be a merely syntactical translation anymore. Instead, due to these functions, the semantics of data rules could be customized and strongly differ from schema rules. In this way, data rules generated from the schema level ones would be much more powerful.

It would be also possible to support two different languages for translations in a framework where the one for instances is a lower level translation of the one for schemas (in the specific case it would imply datalog being translated to SQL for schemas, and straightly SQL for instances). Then the data architect would be allowed to write specific SQL rules for instances whenever the automatically generated ones are not detailed enough.

Adopting an engineered representation of mappings is the most standard solution, since it guarantees a greater expressivity and flexibility. As dealt with in the literature [8], defining an engineered mapping is an open problem and a global approach has not been recognized yet. The main point is that of generating those mappings, hence the implementation of a sufficiently general MATCH operator. Many heuristic algorithms have been proposed in this field, also adopting sophisticated structures for mappings, however their weakness is the absence of an explicit and rich representation of models. This both complicates the process of similarity recognition and leads to model-dependent mappings. MIDST metalevel approach, supporting a complete and extendible meta-representation of models, can lead to simpler definitions of engineered mappings and to more effective matching algorithms. For instance, let us consider the *similarity flooding algorithm* [19,20], an advanced implementation of MATCH. It compares two graphs representing schemas and returns pairs of similar nodes. The similarity of nodes is evaluated both on the basis of heuristic criteria and on a similarity propagation assumption: if two nodes are similar, their neighbors also tend to be similar. Finally a user-aided pruning phase allows to discard the false positives.

The first step of the similarity flooding is the translation of the schemas into graph models. Currently some kind of generality in this field has been achieved by means of *SQL DDL to graph* translating tools [19,20]. Then the nodes are compared with one another independently of the logical function they have in the model of interest. MIDST rich representation of models and schemas would allow for a less syntactical implementation of the algorithm. One would no longer need to pass through the DDL specification of a given database in order to obtain the graph version, since it is already explicitly managed in the metalevel. In addition the graph model over which the algorithm operates could be enhanced with some kind of model-awareness. For example, given a subgraph of the first schema, the algorithm could perform a fake translation in order to isolate the correspondent subgraph in the second schema. Then the similarity flooding comparison might be limited to that portion and, therefore, yield less false positive pairs and involve less user effort.

5 Data Provenance and Quality Problems

The growing number of heterogeneous data together with a uniform access mechanism tend to increase their availability, while inducing a potential decrease in quality. Large and complex workflows involving the integration of pieces of information coming from different data sources present the need for provenance[1] metadata [25]. They enhance data quality under several aspects: possibility to verify whether the data meet the business requirements, establishing creational context, protection of intellectual property and so on.

The shared and uniform access mechanism provided by modern infrastructures makes data prone to losing quality and coherence. Therefore integrity constraints acquire more and more significance in these environments.

A modern approach to these issues should be model-independent and operate at runtime. Whereas there are many proposals for runtime provenance and integrity constraints handlers, to the best of our knowledge, MMS managing this kind of issues model-independently are not currently available.

Here we argue that MIDST metalevel can be extended to cope with these problems with a general approach. It is not worth analyzing possible strategies in detail, however brief explanations should get the idea across.

5.1 Data Provenance

MIDST manages migration of data expressed with directly applicable datalog rules. These rules are coded with respect to MIDST metaconstructs and define manipulations over them. Here it would not be useful to pursue the details of transformation rules, yet some concrete aspects are necessary to outline the main ideas. We adopt a variant of datalog where the unique identifiers (OIDs) of metaconstructs are generated by means of Skolem functions. For a given construct, we define a set of Skolem functions generating OIDs for it from different sets of strongly-typed parameters. These parameters may be other constructs identifiers or constants. Skolem functions are bijective and, for a given construct, the ranges of all the functions defined over it are disjoint. This approach allows to

[1] In the literature *data provenance* is sometimes referred to as *data lineage* or *data pedigree* [25].

easily handle some kind of *where-provenance*. It means that for a given construct occurrence we can track back the whole path leading to it. A construct occurrence is characterized by an OID which is the unique output of a specific Skolem function defined over the construct itself. MIDST handles a global materialization including every function. It is then sufficient to query that materialization to determine the parameters which generated the OID under examination. What is more is that, even though several functions are defined for a given construct, since their ranges are disjoint, one could infer which function has been applied on the basis of the OID value.

A detailed description of the possible implementations of strategies for the data provenance would not be noteworthy here. However the key point is that MIDST handles strongly-typed functions and strictly relates every construct instance to its specific OID. This supports the design of procedures capable of exploring the whole provenance graph. This exploration proceeds as long as a the construct under examination is derived from another construct. When a construct has an OID which derives from the application of a Skolem function to a constant value, the current branch of the exploration terminates.

This idea assumes that MIDST works as a runtime MMS managing the whole net of data migration; in that case pieces of information about *why* and *how provenance* might be useful as well. They convey domain information about the reason why a piece of data is present in a database. In some sense this information is already managed by means of constant parameters of Skolem functions, meaning that they store domain-level notions about the provenance. Anyway an extension to the multilevel dictionary might include a richer description of Skolem functions allowing for an expressive characterization of the *why* and *how* provenance given in a user-chosen language. The power of this approach lays both in the relational representation of metadata and in the strong connection between a metaconstruct and the generating function. While the former aspect is fundamental to guarantee a model-independent handling of data provenance, the second enables an increasingly rich ontological representation of domain information that will be inherently related to the appropriate construct.

So far we have been describing the approach towards data provenance with respect to metaconstructs; in fact MIDST proposes a mechanism allowing for a two-fold definition of transformations: they are valid both for schemas and for instances. Coherently, the platform adopts Skolem functions to define the OIDs of data which are treated alike. So we store a tuple of metadata for each value of each metaconstruct. For example, take the lexical that describes any string-like conceptual element such as entity attributes or table columns. We have one tuple of metadata for each value of every lexical. This representation may seem too fine-grained, yet here it allows for an extremely detailed recording of data provenance based on instance-level Skolem functions that work as we have described for schemas.

In addition, we are not bound to adopt a specific language for transformations, nevertheless it has to guarantee two major points: it must support Skolem functions and have rules acting on metaconstructs. For sake of simplicity we might choose to enrich the SQL language with the possibility of specifying functions for the OID generation. Even its standard version could be used if we interpose a query preprocessing phase applying Skolem functions to generate identifiers.

5.2 Integrity Constraints

In a model management system we may state that handling integrity constraints mainly involves three aspects: definition, application and management. The definition recalls the need to design a language and a metadata representation for constraints; the application is the satisfaction verification, while the management represents the need for an integrated handling in a heterogeneous environment.

It is clear that in a simple context which is usually model-dependent, a well-defined constraint only needs mechanisms for its validation. By contrast, a more complex environment, with translations and migrations among different data models, needs further strategies.

MIDST currently allows for a syntactical definition of both internal and external[2] constraints. As for the first ones, simple *SQL CHECK* conditions are expressed in the metaconstructs by means of their properties. Since the metalevel describes what properties a construct has and potentially their structure, more sophisticated and expressive internal constraints may be defined as well.

External constraints do not even need an explicit definition in terms of properties. For example, we handle a metaconstruct (foreign key) that connects lexicals on the basis of a foreign key constraint. Thus, since lexicals belong to conceptual entities, such as abstracts or aggregations, then the foreign key also models the relationships (and the dependences) between them.

As for the constraint verification, currently MIDST does not offer any effective solution. A syntactical test on instances based on constraints defined over schemas might be implemented, however it would not be the best choice. Integrity constraints allow for an enduring data consistence that must be verified at runtime. Thus, the architectural definition of a more complex MMS should point out what software component is responsible for that verification. Probably, if we consider an MMS as a mediator in a heterogeneous context, the responsibilty for this test will belong to the client database management systems. Instead, if the MMS concentrates the whole responsibility for the data management, it will include a dbms performing integrity checks. Hence an expressive representation of models and effective strategies to import and export constraints should be sufficient for our needs.

A real MMS deals with several models and performs translations among them. One main feature is that of preserving integrity constraints in those transformations. Here the potentialities of MIDST are particularly remarkable since constraints could be managed in a model-independent way. During translations internal integrity constraints are treated as propositional formulas. For example, consider a column of a relational table; in MIDST it is represented by a lexical. Suppose that a *not-nullable* constraint is defined over it. We have a datalog

[2] Constraints are similarly classified as *intra-relational* or *inter-relational* according to the relational terminology.

rule that translates columns into attributes of ER entities. The lexical has the property *isNullable: false* that models the constraint with a simple propositional formula. Normally, before the creation of the new lexical the translation rule does not need to alter the formula that is simply copied. Alternatively, if one wanted to invert that constraint and allow null values in entity attributes, the datalog rule would insert a negation.

In general it means that a datalog rule can involve a complex expression coding the translation of constraints when inter-model translations are performed. Foreign keys work alike. Suppose a foreign key links two lexicals of two different relational tables. When the schema is translated into the ER model, tables turn into entities and specific rules support the translation of integrity constraints into relationships.

This means that a possible direction is using MIDST for constraint definition. Constructs coding them could be defined in the metamodel and would be independent of the specific model. Translations for constraints could be written as well as normal translations currently allowed by the platform and, in a long-term perspective, could be even automatically generated from the metamodel.

6 An Object-Oriented Scenario

In object-oriented applications, the modern approaches towards persistence tend to recognize the benefits of adopting object-to-relational mapping (ORM) strategies [16,18,21,23,27]. Let us briefly define how an application can be considered in terms of schema and instances. Classes define the general structure of the objects: name, attributes and references. Then, the graph of the classes of an application represents its schema. At runtime, on the basis of the definition of the classes, a graph of objects is built. That graph is an instance of the class-based schema.

ORM frameworks are based on the specification of mappings between the object schema (the classes) and the relational schema (the one on the actual database) by means of annotations or other mechanisms involving metadata. In the software development process, the relational database and the software components are not designed at the same time and change independently. These frameworks, implementing a *meet-in-the-middle* approach [11,9], decouple the lifecycles of the two layers by leveraging on mappings. In fact, when the application logic or the database change, it is sufficient to redefine the mappings between them. One research key point in this field is of course the efforts in defining working solutions to the data exchange problem that makes the object instances migrate into the database. Moreover the definition of sophisticated mappings to correctly represent object structures (such as nested classes and hierarchies [10]) is relevant.

An innovative scenario may arise by using MIDST in order to support simple mappings between the object graph and the metalevel. In MIDST metamodel it is possible to determine a set of metaconstructs which represents a complete object-oriented model. Therefore the class schema of an application can be mirrored (imported) into MIDST supermodel and instances can be made to migrate by adopting extremely simple mappings. Unlike traditional ORM, the approach would not handle a static correspondence between two given schemas, instead it would implement a simple import of the object graph into the appropriate subset of the supermodel defining the object-oriented model. It is important to point out that this imports both the schema and the data of the application into MIDST metalevel and treats the result as a normal schema. Therefore model management operations as well as translations can be applied to it.

This introduces a great flexibility allowing for model-independent persistence handling. From another point of view, the described approach represents a sophisticated ORM where instances are memorized in an articulated relational structure (the multilevel dictionary) modelling general object graphs.

What is remarkable is that the explicit representation of schemas leads to a higher level of abstraction. Whereas in ORM we specify correspondences between classes and relations, instance variables and columns, references and foreign keys, here those very concepts are modelled in general in the metalevel and are equally valid for any application.

With traditional ORM only data exchange problems can be used to manage the relationship between the object graph and the database schemas. It also comes as a consequence of the strict pursuing of the meet-in-the-middle approach. In a complete MMS, the adoption of such a strategy would be less tight because of the availability of the whole spectrum of model management solutions.

Besides, MIDST does not only address the meet-in-the-middle approach. If the database is firstly developed (*database-first*), the application logic could benefit from an object graph directly derived from it (export) with a simple (copy) mapping. Conversely, if the application logic is an early activity (*application-logic-first*) in the development process, persistence can be achieved by importing meaningful entities. Finally if meet-in-the-middle is promoted, more sophisticated interrelating mappings can be defined.

Moreover, we could exploit the potentialities of the solutions to model management problems for complex tasks. Advanced relationships between the data model and the program instance itself could be addressed: a change in the data model induces changes in the memory object graph (forward engineering), while objects modifications propagate backwards to the database (round-trip engineering).

Although the illustrated approach has a greater significance in a runtime scenario, from many points of view, an off-line version is meaningful as well. Once an object-oriented graph has been imported into MIDST metalevel, it can be used for a wide variety of targets. First of all, possible applications involve the translation of these schemas into other models not only for persistence reasons, but also to facilitate sharing, integration, migration of data. General purpose model management operators (such as the ones already defined in MIDST) can be then applied to the object schema in the metalevel and modified versions of it can be obtained. The model-independent representation of an object-oriented application could be also used for visualization [15], structural and behavioral analysis.

Examples of specific applications of the structural analysis are anti-pattern detection and dependence verification [26]. On the other hand, behavioral analysis [12] could be based on the fact that an instance obtained from an object graph at a given time is a snapshot of the execution of the process. Different snapshots of an executing program allow to define a track of the execution states that could be related to mappings, handled by means of model management operators and used for debugging reasons.

7 Discussion

This paper, moving from MIDST approach towards schema and data translation, has recalled how the platform can be considered as a general applicator of transformations to schemas. On this basis, reasonings about a possible development of MIDST into a full-featured model management system have been provided. Major problems and research directions for the main challenges have been explained.

References

1. Atzeni, P., Bellomarini, L., Bugiotti, F., Gianforme, G.: A platform for model-independent solutions to model management problems. VLDB Journal (to appear, 2008)
2. Atzeni, P., Cappellari, P., Bernstein, P.A.: A multilevel dictionary for model management. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 160–175. Springer, Heidelberg (2005)
3. Atzeni, P., Cappellari, P., Bernstein, P.A.: Model-independent schema and data translation. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 368–385. Springer, Heidelberg (2006)
4. Atzeni, P., Cappellari, P., Gianforme, G.: MIDST: model independent schema and data translation. In: SIGMOD Conference, pp. 1134–1136. ACM Press, New York (2007)
5. Atzeni, P., Torlone, R.: Management of multiple models in an extensible database design tool. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 79–95. Springer, Heidelberg (1996)
6. Bernstein, P., Haas, L., Jarke, M., Rahm, E., Wiederhold, G.: Panel: Is generic metadata management feasible? In: VLDB, pp. 660–662 (2000)
7. Bernstein, P.A.: Applying model management to classical meta data problems. In: CIDR Conference, pp. 209–220 (2003)
8. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: SIGMOD Conference, pp. 1–12 (2007)
9. Cabibbo, L.: Objects meet relations: On the transparent management of persistent objects. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 429–445. Springer, Heidelberg (2004)
10. Cabibbo, L., Carosi, A.: Managing inheritance hierarchies in object/relational mapping tools. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 135–150. Springer, Heidelberg (2005)

11. Cabibbo, L., Porcelli, R.: M^{2}orm^2: A model for the transparent management of relationally persistent objects. In: Lausen, G., Suciu, D. (eds.) DBPL 2003. LNCS, vol. 2921, pp. 166–178. Springer, Heidelberg (2004)
12. Giguette, R., Hassell, J.: A relational database model of program execution and software components. In: ACM-SE 38: Proceedings of the 38th annual on Southeast regional conference, pp. 146–155. ACM Press, New York (2000)
13. Haas, L.M.: Beauty and the beast: The theory and practice of information integration. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 28–43. Springer, Heidelberg (2006)
14. Halevy, A.Y., Ashish, N., Bitton, D., Carey, M.J., Draper, D., Pollock, J., Rosenthal, A., Sikka, V.: Enterprise information integration: successes, challenges and controversies. In: SIGMOD Conference, pp. 778–787 (2005)
15. Hamer, J.: Visualising java data structures as graphs. In: ACE 2004: Proceedings of the sixth conference on Australasian computing education, pp. 125–129 (2004)
16. Hibernate, `http://www.hibernate.org/`
17. Hull, R., King, R.: Semantic database modelling: Survey, applications and research issues. ACM Computing Surveys 19(3), 201–260 (1987)
18. Java Data Objects, `http://www.jdocentral.com/`
19. Melnik, S.: Model management: First steps and beyond. In: BTW, pp. 455–464 (2005)
20. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: ICDE, pp. 117–128 (2002)
21. Microsoft ObjectSpaces, `http://msdn.microsoft.com/library/default.asp?url=/li-brary/en-us/dnadonet/html/objectspaces.asp/`
22. Mork, P., Bernstein, P., Melnik, S.: A schema translator that produces object-to-relational views. Technical Report MSR-TR-2007-36, Microsoft Research (2007), `http://research.microsoft.com`
23. Oracle AS TopLink, `http://otn.oracle.com/products/ias/toplink/`
24. Papotti, P., Torlone, R.: Heterogeneous data translation through XML conversion. J. Web Eng. 4(3), 189–204 (2005)
25. Simmhan, Y., Plale, B., Gannon, D.: A survey of data provenance in e-science. In: ACM SIGMOD International Conf. on Management of Data, vol. 34(3), pp. 31–36 (2005)
26. Structural Analysis for Java, `http://www.alphaworks.ibm.com/tech/sa4j/`
27. The Java Persistence API - A Simpler Programming Model for Entity Persistence, `http://java.sun.com/developer/technicalarticles/j2ee/jpa/`