MQL: A Mapping Management Language for Model-based Databases

Valéry Téguiak[1], Yamine Ait-Ameur[2], Stéphane Jean[1] and Éric Sardet[3]
[1]*LIAS, ISAE-ENSMA and Poitiers University, Futuroscope, Poitiers, France*
[2]*IRIT-ENSEEIHT, INPT-ENSEEIHT, Toulouse, France*
[3]*CRITT Informatique, Futuroscope, Poitiers, France*

Keywords: Mapping, Meta-modeling, Model Transformation, Ontology Engineering, Query Languages.

Abstract: Nowadays model mapping plays a crucial role in applications manipulating various heterogeneous sources (data integration and exchange, datawarehouse, etc.). Users need to query a given data source and still obtain results from other mapped sources. If many model management systems have been proposed that support high-level operators on model mappings, a more flexible approach is needed supporting the querying of mapping models and the propagation of queries through mappings. As a solution, we present, in this paper, a mapping-based query language called MQL (Mapping Query Language). MQL extends the SQL language with new operators to exploit mappings. We show the interest of this language for the multi-model ontology design methodology proposed in the DaFOE4App (Differential and Formal Ontology Editor for Application) project.

1 INTRODUCTION

In order to deal with various heterogeneous models used to represent the same real word domain, several mapping languages (Bouquet et al., 2003; Horrocks et al., 2004) or frameworks (Jouault et al., 2008; Melnik et al., 2003; Moha et al., 2010) have been proposed. These frameworks support either model mappings or model transformations. (Bouquet et al., 2003; Horrocks et al., 2004) allow users to express correspondences between models and (Jouault et al., 2008; Melnik et al., 2003; Moha et al., 2010) describe model transformations. Both approaches aim at performing instance migration. Most of these languages run in central memory and do not address scalability when dealing with huge amount of data.

Moreover, with the emergence of the Web, the amount of models and instances is growing drastically. Managing mappings in such a context often requires writing more and more undesirable complex queries. Therefore, offering solutions for managing such mappings and instances in a convenient way becomes a necessity if one wants to address real sized problems.

Before year 2000, mappings were implemented by programs, then (Bernstein, 2003) introduced the notion of *Model Management* that aimed at reducing the amount of programming needed for the development of metadata-intensive applications. More precisely, (Bernstein, 2003) has provided model management operators (e.g, *compose, diff, merge, match, etc*) allowing to manipulate and to manage models and mappings as objects. However, to understand and to use mappings established between source models, designers need to query and to exploit them in order to express a query on a data source and to obtain data results from other sources. Thus, a more flexible approach is needed for supporting the querying of mapping model and the propagation of queries through mappings. As a solution, we propose in this paper a mapping-based query language named MQL (Mapping Query Language). This language is an extension of traditional SQL query language with new operators to exploit mappings such as crossing or filtering mappings. The interest of this language is shown on a real use case extracted from the DaFOE4App project.

This paper is organized as follows. Section 2 describes the use case set up to show the interest of our proposition. This use case is an ontology design methodology based on a multi-models approach. Section 3 discusses related work. After presenting our requirements for a new query language in Section 4, we present, in Sections 5 and 5.1, our MQL language proposal. Finally, Section 6 concludes this paper and gives some perspectives of this work.

Téguiak V., Ait-Ameur Y., Jean S. and Sardet É..
MQL: A Mapping Management Language for Model-based Databases.
DOI: 10.5220/0003998601450150
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 145-150
ISBN: 978-989-8565-10-5
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

2 CASE STUDY

In this section, we describe the ontologies design process led by the DaFOE platform (a demonstration of this platform is available at http://testcritt.ensma.fr/dafoe/demo/dafoeV1.zip), where our MQL language proposal has been applied. This platform proposes a stepwise approach for building an ontology starting from text.

2.1 Ontology Design in the DaFOE

The DaFOE platform provides a stepwise methodology for building ontologies from text analysis. The first step is dedicated to linguistic analysis (Terminology step) in which users manage linguistic information (terms and relations between terms) extracted with natural language processing tools. Then, a step for terms disambiguation (TerminoOntology step) is performed. Finally, a formalization step (Ontology step) allows users to create *classes* and *properties* of the ontologies. Each step, that is autonomous, has its own model respectively presented in Figure 1, 2 and 3 and mappings are used to establish correspondences between these models.

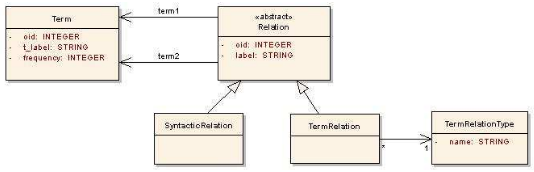


Figure 1: A subset of the Terminology model.

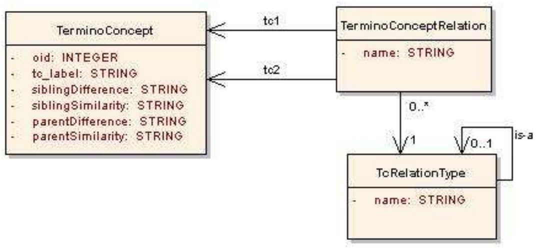


Figure 2: A subset of the TerminoOntology model.

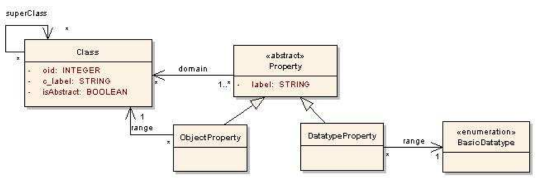


Figure 3: A subset of the Ontology model.

2.2 Persistence of Mappings

In (Téguiak et al., 2012), we argued that model-based databases (MBDB) are well adapted for handling mappings in a database context. In that proposal, we have extended MBDB with a repository for mapping representations as illustrated in Figure 4. In the resulting meta-model (named core meta-model) where models are defined by their entities and their attributes, three main constructors for creating correspondences are available. The first one, called *mLink*, is used to establish correspondences between models. The second one, called *eLink*, allows the user to establish correspondences between entities of models and finally, the *aLink* uses an *expression* to write the target attribute in term of the sources attributes.

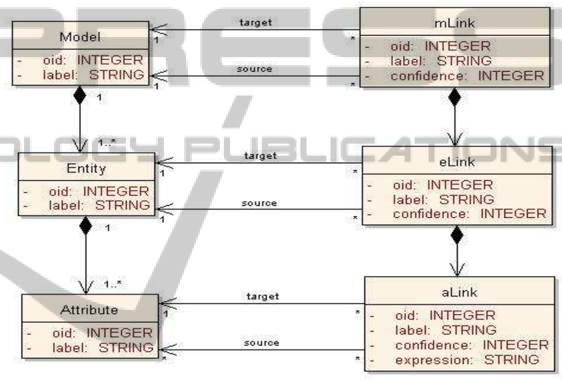


Figure 4: Core metamodel.

2.2.1 Terminology to TerminoOntology Step

Considering both *Terminology* and *TerminoOntology* models, a simplified mapping between these models consists in:

- Creating a *mLink* between the *Terminology* model and the *TerminoOntology* model;

- Creating a *eLink* from the *Term* entity and the *TerminoConcept* entity to express that instances of the *Term* entity will be transformed into instances of the *TerminoConcept* entity;

- Creating a *aLink* expressing that an instance of the *TerminoConcept* entity has the same *label* as the one of its corresponding instances of the *Term* entity, prefixed by *'tc_'*. Another *aLink* expresses that the *rate* of an instance of *TerminoConcept* entity, equals to the *frequency* of the corresponding instance in the *Term* entity divided by 100.

2.2.2 TerminoOntology to Ontology Step

For *TerminoOntology* and *Ontology* models, a simplified mapping consists in:

- Creation a *mLink* between the *TerminoOntology* model and the *Ontology* model;

- In the context of the previous created *mLink* between models, a *eLink* is created between the *TerminoConcept* entity and the *Class* entity to express that instances of the *TerminoConcept* entity will be transformed into instances of the *Class* entity;

- Creating of a *aLink* expressing that an instance of the *Class* entity has the same *label* as the one of its corresponding instance in the *TerminoConcept* entity. Another *aLink* expresses that the *relevance factor* of an instance of *Class* entity, equals to the *rate* of the corresponding instance in the *TerminoConcept* divided by 10.

As an illustration, assume that instances of the Ontology, TerminoOntology and Terminology models are represented by Tables 1, 2 and 3 respectively. Thanks to mappings, a user who queries the *Class* entity of the Ontology model could want to query both TerminoConcept of the TerminoOntology model and Term of Terminology model.

Table 1: Ontology model.

Class			
oid	c_label	relevance	isAbtract
1000	tc_car	0.01	true
1001	tc_wheel	0.002	false
1003	electric_motor	0.04	false

Table 2: TerminoOntology model.

TerminoConcept		
oid	tc_label	rate
600	tc_car	0.1
602	motor	0.08
603	motorcycle	0.8

Table 3: Terminology model.

Term		
oid	t_label	frequency
300	car	1
301	wheel	2
302	bicycle	30

Putting these mappings all together results in the MOF-like database repository (Cf. Figure 5) where M_{i+1}/M_i means that the M_i level is represented as instance of the M_{i+1} level. The meta-schema part is dedicated for managing the core metamodel while schema and instance part a dedicated for managing business models and data respectively.

3 RELATED WORK

Metadata repository systems manage metadata commonly represented as models or meta-models. Such a repository is often equipped with a MOF-based query language ((Lakshmanan et al., 2001), MSQL (Grant et al., 1993), SQL/M (Kelley et al., 1995), OntoQL (Jean et al., 2006), mSQL (Petrov and Nemes, 2008), SparQL (Konstantinos et al., 2010)) that provides capabilities to manipulate both data and meta-data.

Another query language, called mapping oriented query language ((Melnik et al., 2003), (Konstantinos et al., 2010)) provide and explicit representation of mappings between models and offer capabilities to exploit these mappings when querying data. As limitation, these languages do not allow a user to customize the mapping exploitation process. In many cases, the exploitation process is hidden to the user and all the graph of interconnected database is used even if the user wants to use only a sub-part of this graph. Furthermore, in these languages or frameworks, the representation of mappings is static and can not be extended dynamically.

As illustrated in our case study, our proposed database structure (Cf. Figure 5) is a MOF-like database that also handles mappings between models. However, as we will see in the next section, this database is a bit more complex to manage using classical SQL queries. This drawback brings us to design a query language bypassing limitations of languages presented above and that makes easier mappings exploitation using high level operators. So, requirements for such a language are needed.

4 REQUIREMENTS

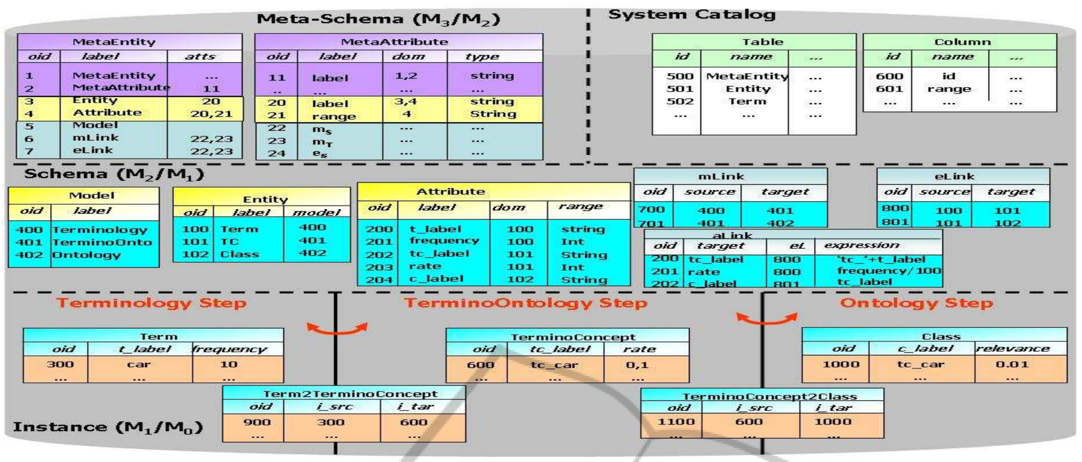
(Wakeman and Jowett, 1993; Petrov and Nemes, 2008) have investigated requirements for higher-level query languages managing both data and metadata (e.g, models). In this section we introduce new requirements specific to mappings exploitation.

4.1 Handling Complex Queries

Considering the example of Section 2 and assume that a user wants to retrieve, for the ontology model, all classes of the ontology model whose relevance factor is high than 0.01. To achieve this goal, the user can write the following query:

R$_1$) SELECT c_label, relevance FROM Class WHERE relevance $\geq$ 0.01.

However, if the user also want to retrieve, for each class, the corresponding object in other models



MetaEntity

oid	label	atts
1	MetaEntity	...
2	MetaAttribute	11
3	Entity	20
4	Attribute	20,21
5	Model	
6	mLink	22,23
7	eLink	22,23

MetaAttribute

oid	label	dom	type
11	label	1,2	string
..	...	...	...
20	label	3,4	string
21	range	4	String
22	m_S	...	...
23	m_T	...	...
24	e_S	...	...

Table

id	name	...
500	MetaEntity	...
501	Entity	...
502	Term	...
...	...	...

Column

id	name	...
600	id	...
601	range	...
...	...	...

Model

oid	label
400	Terminology
401	TerminoOnto
402	Ontology

Entity

oid	label	model
100	Term	400
101	TC	401
102	Class	402

Attribute

oid	label	dom	range
200	t_label	100	string
201	frequency	100	Int
202	tc_label	101	String
203	rate	101	Int
204	c_label	102	String

mLink

oid	source	target
700	400	401
701	401	402

eLink

oid	source	target
800	100	101
801	101	102

aLink

oid	target	eL	expression
200	tc_label	800	'tc_'+t_label
201	rate	800	frequency/100
202	c_label	801	tc_label

Term

oid	t_label	frequency
300	car	10
...	...	...

TerminoConcept

oid	tc_label	rate
600	tc_car	0,1
...	...	...

Class

oid	c_label	relevance
1000	tc_car	0.01
...	...	...

Term2TerminoConcept

oid	i_src	i_tar
900	300	600
...	...	...

TerminoConcept2Class

oid	i_src	i_tar
1100	600	1000
...	...	...

Figure 5: Mapping management in the DaFOEApp project.

mapped to the ontology model, two situations may occur.

On the one hand, if the user knows the mappings characteristics, so he/she can manually write the appropriate following SQL queries:

R$_2$) SELECT tc_label, rate FROM TerminoConcept WHERE rate/10 $\geq$ 0.01

R$_3$) SELECT t_label, frequency FROM Term WHERE (frequency/100)/10 $\geq$ 0.01

R_2 and R_3 queries are translation of the R_1 query on the TerminoOntology and Terminology models respectively according to the mappings between these models.

On the other hand, because mappings characteristics may be evolved dynamically (new mappings may be created while existing one may be deleted or updated, just as in a peer to peer system (Iraklis and Joemon, 2003)), one needs firstly to query mappings repository for characteristics retrieval, and then write the appropriate queries based on these characteristics. Such a query requires to access a repository which represented instances are model and mappings between models. According to the transitivity capability of mappings, this access may raise a syntactic complex query (Cf. Table 4). To simplify, we assume that additional data should be retrieved from the TerminoOntology model.

Table 5 represents the results of the Q_5 query. These results are exploited to generate, for the TerminoOntology model, the query for retrieving data.

As we can observe, the process of unfolding queries on target models is not easy and may become complex if one needs to integrate the complete network of mappings. In this case, the user handles by himself the transitivity capabilities of mappings. A classical approach to deal with this situation consists in writing a query translator. So, the user writes a query (R_1 query for example) and the translator generates queries for target models. This queries generation process is hidden to the user and made implicit. In other words, this approach assumes that the user does not know any mappings characteristics usable to customize the queries generation process.

Table 4: Mapping level queries.

Goals	**Queries**
Q_1) *Retrieve the Ontology model.*	SELECT M.oid FROM Entity E, Model M WHERE E.label= "Class" AND E.model= M.oid
Q_2) *Retrieve mLink where the Ontology model is involved as target*	SELECT mLink.oid FROM mLink WHERE mLink.target in Q_1
Q_3) *Retrieve the entities mapped to Class entity.*	SELECT eLink.source FROM eLink, Entity E WHERE eLink.mL in (Q_2) AND eLink.oid= E.oid AND E.label= "Class"
Q_4) *Retrieve correspondences between entities where the Class entity is involved as target*	SELECT eLink.oid FROM eLink WHERE eLink.mL in (Q_2)
Q_5) *Retrieve mapped entities and mapped attributes (through their expression).*	SELECT E.label,aLink.exp FROM Entity E, Attribute A, aLink WHERE E.oid in (Q_3) AND aLink.eL in (Q_4) AND aLink.target= A.oid AND A.dom= E.oid

Table 5: Mapping level results.

E.label	**aLink.expression**
TerminoConcept	tc_label
TerminoConcept	rate/10

4.2 Handling Mapping Navigation

Considering more closely the second situation of the requirement presented in Section 4.1 where users need to query mappings repository in order to retrieve mappings characteristics. One can ask itself how to handle transitivity with query languages such as SQL for example. This issue refers to the needs to dynamically navigate through the mappings hiding the exploitation of these mappings. So, a policy for a transitive subqueries propagation through chains of arbitrarily huge mapped models is required because these models may contain huge amount of data.

4.3 Providing Persistent Mappings

This requirement refers to the problem of *memory saturation*, that means avoiding loading into central memory big amount of data whose models are mapped together. Indeed, the mappings repository may become very huge and therefore expensive (in response time and memory consumption) for navigation purposes because, new models (says news modeling steps) could be created dynamically according to the needs of a particular user. Thus, a persistent-based approach is required.

5 OUR APPROACH

In this section, we present an overview of the MQL (Mapping Query Language), our mapping-based query language proposal for handling mappings according to previous quoted requirements. This language is highly coupled to the Model Based Database (MBDB) persistence approach presented in (Téguiak et al., 2012). For each part (meta-schema, schema, instance) of the MBDB, the MQL language provides operators to define, manipulate and query its content. Due to space limitation, we only present capibilities for MQL to query instances and mappings together. More details are available in a complete version of our unpublished internal report (Téguiak et al., 2011).

5.1 Instances and Mappings Together

To address the requirements mentioned in Section 4, we propose to extend the classical "SELECT ... FROM ... WHERE ... " query. In other words, our approach is and hybrid one that can be used even if a user knows mappings characteristics or not. As the main purpose of MQL is to facilitate navigation through mappings, we introduced optional statements useful for query propagation in order to get compact syntactic queries. The following statements are exploited in the queries translation process to customize this process.

MATCH. Specify the target models in which the MQL query is propagated at runtime.

FILTER. When propagating a MQL query from a model m_1 to another model m_2, an entity of m_1 may correspond to several entities of m_2. In this case, one may want to restrict the translation so that it applies only to part of these entities. Such a restriction is described using the FILTER clause.

CONFIDENCE. Confidence degrees are often assigned to mappings in order to handle fuzzy mappings. This clause restricts the propagation of the MQL query for the models that satisfy the specified confidence degree. When specified, this clause is used as a threshold to be respected.

With closure. If specified, the propagation of the query is achieved through the mappings repository using the *transitive closure* in the way that, instances are retrieved according to the transitivity of available mappings.

DEPTH. When a MQL query uses the *With closure* clause, it may result in a memory saturation or a bad response time according to the size of the graph of mappings. The DEPTH clause specifies the depth exploration of the graph of mappings. For example, *"DEPTH 4"* means that the MQL query will be propagated transitively on four consecutive mappings at most .

mWHERE. Unlike the classical WHERE clause of a SQL query, the mWHERE clause allows users to specify predicates to filter correspondences. In other words, the mWHERE clause is comparable to a SQL WHERE clause, but it is dedicated to mapping level.

5.2 MQL in Action

Applied to the Ontology model, the mQ_1 query returns data (Cf. Table 6) of the Ontology model (no mapping statement is used). In other words, this query is a classical SQL query. For readability purpose, all the result records are prefixed by the name of its entity.

Table 6: Results of the mQ_1 MQL query.

mQ_1	Results

SELECT c_label, relevance
FROM Class
WHERE relevance $\geq$ 0.01
	Class(tc_car, 0.01)
	Class(electric_motor, 0.04)
	...

Applied to the Ontology model, the mQ_2 query returns data (Cf. Table 7) extracted from both the Ontology and the TerminoOntology models (the

MATCH statement has been set to TerminoOntology).

Table 7: Results of the mQ_2 MQL query.

mQ_2	Results

SELECT c_label, relevance
FROM Class
WHERE relevance $\geq$ 0.01
MATCH TerminoOntology
	Class(tc_car, 0.01)
	Class(electric_motor, 0.04)
	TC(motorcycle, 0.8)
	...

Applied to the Ontology model, the mQ_3 query returns data (Cf. Table 8) extracted from both Ontology and TerminoOntology models (the MATCH statement for this query has been set to all models using the * symbol). However due to the DEPTH statement, the results are limited to 1 transitive propagation. Only the TerminoOntology model is reachable from the Ontology model with 1 propagation.

Table 8: Results of the mQ_3 MQL query.

mQ_3	Results

SELECT c_label, relevance
FROM Class
WHERE relevance $\geq$ 0.01
MATCH *
FILTER *
DEPTH 1
With closure
	Class(tc_car, 0.01)
	Class(electric_motor, 0.04)
	TC(motorcycle, 0.8)
	...

Applied to the Ontology model, the mQ_4 query returns data (Cf. Table 9) extracted from both the Ontology, TerminoOntology and Terminology models. Indeed, thanks to the * symbol of the MATCH statement and with no DEPTH limitation, mQ_4 is propagated to any model transitively reachable from the Ontology model.

Table 9: Results of the mQ_4 MQL query.

mQ_4	Results

SELECT c_label, relevance
FROM Class
WHERE relevance $\geq$ 0.01
MATCH *
FILTER *
With closure
	Class(tc_car, 0.01)
	Class(electric_motor, 0.04)
	TC(motorcycle, 0.8)
	Term(bicycle, 30)
	...

6 CONCLUSIONS

In this paper, we have presented a mapping-based query language called MQL that makes easier querying data thanks to available mappings between models. This language has a knowledge part based on a core metamodel dedicated models and mappings representation. One of the main features of our approach is that this knowledge part can be extended by evolving the core metamodel. MQL has been implemented for model-based databases, where both instance, metamodel and metametamodel level are persisted in a single database. As perspective of this work, we are working on the definition of a benchmarking scenario for improving performance of our approach.

REFERENCES

Bernstein, P. A. (2003). Applying model management to classical meta data problems. In *CIDR*.

Bouquet, P., Giunchiglia, F., Harmelen, F. V., Serafini, L., and Stuckenschmidt, H. (2003). C-owl: Contextualizing ontologies. In *ACM SIGIR'03*, pages 164–179.

Grant, J., Litwin, W., Roussopoulos, N., and Sellis, T. (1993). Query languages for relational multidatabases. In *VLDB*.

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean., M. (2004). Swrl: a semantic web rule language combining owl and ruleml.

Iraklis, K. and Joemon, J. (2003). An architecture for peer-to-peer information retrieval. In *ACM SIGIR'03*, pages 401–402.

Jean, S., Ait-Ameur, Y., and Pierra, G. (2006). Querying ontology based database. the ontoql proposal. In *SEKE*, pages 166–171.

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). Atl: a model transformation tool. In *Science of Computer Programming*, pages 31–39.

Kelley, W., Gala, S., Kim, W., Reyes, T., and Graham, B. (1995). Schema architecture of the unisql/m multidatabase system. In *Modern Database Systems*.

Konstantinos, M., Nektarios, G., Nikos, B., and Stavros, C. (2010). Ontology mapping and sparql rewriting for querying federated rdf data sources. In *OnTheMove*, pages 1108–1117.

Lakshmanan, L., Sadri, F., and Subramanian, S. N. (2001). Schemasql: An extension to sql for multidatabase interoperability. In *JTDS*.

Melnik, S., Rahm, E., and Bernstein, P. A. (2003). Developing metadata-intensive applications with rondo. In *Journal of Semantic Web*, pages 47–74.

Moha, N., Sen, S., Faucher, C., Barais, O., and Jézéquel, J.-M. (2010). Evaluation of kermeta for solving graph-based problems. In *JSTT*.

Petrov, I. and Nemes, G. (2008). A query language for mof repository systems. In *OnTheMove*, pages 354–373.

Téguiak, V., Ait-Ameur, Y., and Sardet, E. (2012). Use of persistent meta-modeling systems to handle mappings for ontology design. In *MOPAS*, page To appear.

Téguiak, V., Ait-Ameur, Y., Sardet, E., and Bellatreche, L. (2011). MQL: an extension of SQL for mappings manipulation. Technical report, LIAS.

Wakeman, L. and Jowett, J. (1993). *PCTE: the standard for open repositories*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.