A generic query-translation framework for a mediator architecture

Jacques Calmet Sebastian Jekutsch Joachim Schü

Department of Computer Science
Institute for Algorithms and Cognitive Systems
University of Karlsruhe
{calmet,jekutsch,schue}@ira.uka.de

Abstract

A mediator is a domain-specific tool to support uniform access to multiple heterogeneous information sources and to abstract and combine data from different but related databases to gain new information. This middleware product is urgently needed for these frequently occurring tasks in a decision support environment. In order to provide a front end, a mediator usually defines a new language. If an application or a user submits a question to the mediator, it has to be decomposed into several queries to the underlying information sources. Since these sources can only be accessed using their own query language, a query translator is needed.

This paper presents a new approach for implementing query translators. It supports conjunctive queries as well as negation. Care is taken to enable information sources of which processing capabilities do not allow conjunctive queries in general. Rapid implementation is guided by reusing previously prepared code. The specification of the translator is done declaratively and domain–independent.

1 Introduction

Decision support systems rely heavily on existing information sources like databases containing financial or engineering data, specialized expert systems or the world wide web, just to mention a few. Whether an executive needs to access different sources in his daily work or a computer system automates this procedure: a tool is needed to facilitate the utilization of these heterogeneous, independently maintained, and autonomous information sources. Mediators are domain-specific integration modules which build up an intermediate layer between the underlying information sources and the applications using them. They "simplify, abstract, reduce, merge and explain data" to "create information for a higher layer of applications" [22]. They do not just overcome platform mismatches but also contain a knowledge base which describes how relevant new information is acquired using the information sources.

Different approaches for developing mediators have been presented, for example [17, 21, 8, 15]. The work described in this paper is based on the Karlsruhe Open MEdiator Technology (KOMET) project [6]. In KOMET the mediatory knowledge base is written in a declarative manner using annotated logic [16]. The language, called KAMEL (KArlsruhe MEdiator Language), facilitates typical mediation tasks like integrating incompatible schemas [7], preferring more reliable sources, making temporal inferences or joining data from different sources to deduce new information.

This paper introduces a new approach for translating queries expressed in KAMEL to equivalent queries expressed in the native query language of an arbitrary information source. It is described what needs to be considered to support different types of sources and how a query translator can be implemented rapidly.

2 KOMET and query translation

We will give only a very short introduction to the concepts behind KAMEL. For a much deeper treatment and more sophisticated examples, the reader may refer to [16, 20, 14, 6].

KAMEL is a logic programming language based on generalized annotated logic [12, 16]. A mediator program consists of a set of annotated clauses of the form

$$A : \nu \leftarrow C_1 \,\&\, \ldots \,\&\, C_n \;||\; B_1 : \mu_1 \,\&\, \ldots \,\&\, B_k : \mu_k .$$

The *head* $A : \nu$ is an annotated atom where A is a usual formula of datalog and ν is an annotation. The $B_i : \mu_i$ are annotated literals. They make up the *body*

of the clause. Finally, in the *constraint part* the C_i are ordinary datalog literals.

If I is an interpretation of the program, then $I \models A : \mu$ iff $I(A) \succeq \mu$, i.e. $A : \mu$ holds if the truth value of A is equal to or higher than μ. μ is defined on a lattice of values, for example truth values (linear or non-linear), time information, uncertainty and fuzzy values, etc. [12].

The literals in the constraint part denote calls to the information sources. Note that no annotations appear in the constraint part, so they will not be treated any further in this paper. A constraint literal is preceded by a source specifier. Consider as an example the following short mediator program:

$$
\begin{aligned}
stock(Name, Close) : [\{Date\}] \quad \leftarrow \\
\qquad DB :: \{ & stockname(Name, SID) \ \& \\
& close(SID, Date, Close) \ \} \ || \\
stock(Name, Close) : [\{today\}] \quad \leftarrow \\
\qquad T :: & stock(Name, Close) \ || \\
equal\_closing(Name1, Name2) : [\{Date\}] \quad \leftarrow \\
\qquad || \ & stock(Name1, Close) : [\{Date\}] \ \& \\
& stock(Name2, Close) : [\{Date\}] \ \& \\
& Name1 \neq Name2.
\end{aligned}
$$

In the first clause a relation between the full name of a stock and its closing price is defined using the information in the source called DB. DB is an SQL-database with relations *stockname* and *close*. The derived data in *stock* is annotated with the time of trade. The second clause extracts the same information for the actual time from a ticker T. This ticker provides the full name of each stock combined with its actual price. *today* is a constant. The third clause uses the *stock* predicate to extract those pairs of stock which have the same closing price for a given date. In general there is no limit on the number of different sources addressable in a clause. Since query translation is an operation done for each information source independently, this example will suffice.

Suppose the question $\leftarrow stock(Name, 500.00) : [\{Date\}]$ has been submitted to the mediator. The goal-oriented evaluation procedure of KOMET [14] (see [5] for a bottom-up procedure) will send the subgoal (query 1):

$$
\begin{aligned}
ans(Name, Date) \quad \leftarrow \quad & stockname(Name, SID) \ \& \\
& close(SID, Date, 500.00) \quad (1)
\end{aligned}
$$

to the query translator for DB. The body is partly instantiated. The head is a pseudo-predicate *ans* containing all the variables which values are requested by the mediator. Note that the values of SID are not needed in the mediator. For DB this query should be translated to

```
select r1.name, r2.date
from stockname r1, close r2
where r1.sid = r2.sid and r2.value = 500.00
```

Representing the constraint part as a conjunction of ordinary literals makes it possible to collect and combine subqueries directed to one and the same information source. For example, if $\leftarrow equal\_closing(`SAP`, Name) : [\{Date\}]$ is sent to the mediator, DB gets the query

$$
\begin{aligned}
ans(Name, Date) \quad \leftarrow \quad & stockname(`SAP`, SID1) \ \& \\
& stockname(Name, SID2) \ \& \\
& close(SID1, Date, C) \ \& \\
& close(SID2, Date, C).
\end{aligned}
$$

This query can be sent to DB and translated as a whole. This ensures that most of the computation is done by the information source, which is specialized for this task in contrast to the mediator.

The translator as well as the mediator need to know the predicates and their signature of every information source, for example $stockname(string, number)$ and $close(number, float, time)$ in DB. This collection is called the *export schema* of an information source.

In the sequel we describe a framework for query translator production which provides the following features:

- It supports conjunctive queries (or SPJ-queries), not only isolated literals.
- It supports negation.
- It supports information sources which do not have the processing capabilities to answer conjunctive queries in general. In this case the translator divides a conjunctive query into a minimal set of single queries and combines the partial answers to the demanded one. This happens without consulting the mediator.
- It allows rapid implementation of translators guided by the weaknesses of the information source.
- It has an architecture which supports the reuse of previously implemented parts. For example, if a generic SQL-translator has been implemented, only a small amount of work has to be done to customize it for specific DBMS like Informix or Oracle.

- It is domain-independent, i.e. the same Oracle-translator code can be used for different databases (export schemas) running on Oracle-DBMS.

The translation is done in two steps:

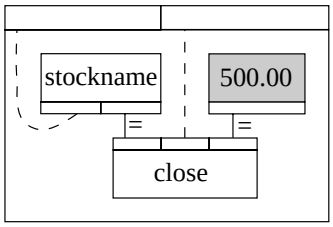
1. The conjunctive query is transformed into an intermediate language called "box representation". This representation is best suited for the second step. Section 3 describes this transformation and defines the semantics of the box representation.

2. The box representation of a query is translated via production rules into a query in the native language. There is a fixed set of rules each of which may be disabled according to the processing capabilities of the information source. Section 4 addresses this translation scheme.

Section 5 is devoted to the software engineering aspect of how to implement a query translator using a library of routines and classes (framework). The paper closes with a comparison to related approaches.

3 Box representation of conjunctive queries

The box representation of query 1 is the following:



It consists of two white boxes, called *N-boxes*, denoting the two atoms of the query, and a dark shaded box, called *M-box*, denoting a constant occurring in the query. Every box has attached *docks*, which represent the arguments of the atoms. A labeled *arc* connects two docks if they are represented by the same variable in the query. The constant 500.00 is separated as an M-box with an arc connected to the dock where it originally occurred. A *C-box* is wrapped around this construct with dashed arcs (*projection-arcs*) representing the values to be exported. A more formal definition of the syntax and semantics of the box representation follows.

Syntax A box representation consists of four types of boxes each with a different attribute: *N-boxes* (white) have a name, *A-boxes* (shaded) a query-string, *M-boxes* (dark shaded) a relation (i.e. a set of value tuples) and *C-boxes* a box representation as an attribute. Each box B contains an ordered row $d(B)$ of docks referenced by their indices. There are four types of arcs: A *join-arc* connects two docks of different boxes, a *selection-arc* connects two docks of the same box, a *box-arc* connects two boxes directly and a *projection-arc* (dashed) connects a dock of a C-box with a dock of a box inside this C-box. Arcs of the first three types are labeled. Arcs are directed to support the semantics of the labels.

Semantics The semantics of the box representation can be defined in terms of the relational algebra. See for example [1] for the definitions of the usual unnamed relational operators $\sigma, \pi, \bowtie$. A *sort* is a set of values of the same type, for example *string*. A *relation descriptor* R is a name together with a row of sorts. A *relation* I_R is a set of tuples over a Cartesian product of the sorts denoted by the descriptor R. For example $R = stock(string, float)$ and $I_R = \{\langle `SAP`, 500.00\rangle, \langle `BMWHold.`, 257.30\rangle\}$. I can be seen as an instantiation of a descriptor. The *export schema* of an information source consists of a set of descriptors. An *information source* is an instantiation of each descriptor in its export schema.

Definition 3.1 The *reduction* $red(B, I)$ (or simply $red(B)$) of a box representation B over information source I is defined as follows:

(1) If B is an N-box with relation descriptor R as the attribute, then $red(B) = I_R$.

(2) If B is an M-box with relation T as the attribute, then $red(B) = T$.

(3) If B is an A-box with a formula F of the relational algebra as the attribute, then $red(B) = F$.

(4) If B is a C-box, then $red(B) = red(B')$, where B' results from B by means of application of one of the following rules. At least one of these rules is applicable because of the syntax of the box representation.

(4.1) If B has a box-arc A between boxes B_1 and B_2 labeled with θ, then replace A, B_1, B_2 by a new A-box which obtains the attribute $red(B_1)\ \theta\ red(B_2)$ and the docks $(d(B_1), d(B_2))$[1].

[1] (d_1, d_2) denotes the concatenation of the elements in d_1 and d_2 into one row. $(d_1, d_2, d_3, \ldots, d_n) := ((\ldots((d_1, d_2), d_3), \ldots), d_n)$.

(4.2) If B has a join-arc A between docks d_1 (at box B_1) and d_2 (at box B_2) labeled with θ, then replace A, B_1, B_2 by a new A-box which obtains the attribute $red(B_1) \bowtie_{d_1 \theta d_2} red(B_2)$ and the docks $(d(B_1), d(B_2))$.

(4.3) If B has a selection-arc A between docks d_1 and d_2 of box B_1 labeled with θ, then replace A, B_1 by a new A-box which obtains the attribute $\sigma_{d_1 \theta d_2}(red(B_1))$ and the docks $d(B_1)$.

(4.4) If B contains the boxes $B_1, \ldots, B_m$ and no arcs but projection-arcs, then B' is the A-box with the attribute $\pi_{P(B_1)}(B_1) \times \cdots \times \pi_{P(B_m)}(B_m)$ and the docks $(P(B_1), \ldots, P(B_m))$ where $P(B_i)$ is the row of docks of B_i which are connected with a projection-arc.

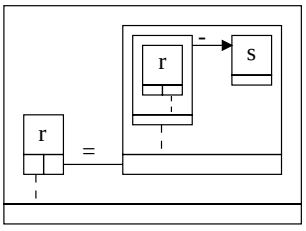
For example, the box representation B depicted above has the reduction $red(B) = \pi_{1,4}(stockname \bowtie_{2=1} (close \bowtie_{3=1} \{\langle 500.00 \rangle\}))$. The rules mentioned will be revisited in section 4.

The algorithm of how to transform a conjunctive query into a box representation is straightforward. It is based on the fact that every conjunctive query has a normal form with explicit equations and without constants. For example, query 1 may be written as:

$$\begin{aligned} ans(X_1, X_2) \quad \leftarrow \quad & stocknames(X_1, Y_1) \ \& \\ & close(Y_2, X_2, Z_1) \ \& \ Y_1 = Y_2 \ \& \\ & \{\langle 500.00 \rangle\}(Z_2) \ \& \ Z_1 = Z_2. \end{aligned}$$

Here $\{\langle 500.00 \rangle\}$ represents a predicate which is true for exactly every tuple the name denotes, in this case only for $\langle 500.00 \rangle$. The interested reader may refer to [11] for the algorithm and its correctness proof.

It is also possible to represent negated literals in the box representation, although this results in slightly more complex constructs. Instead of usually inserting an N-box for an atom, a negated atom is represented by a C-box. An abstract example may suffice: $ans(X) \leftarrow r(X, Y) \ \& \ \neg s(Y)$ is transformed into the following box representation B:



Here $red(B) = \pi_1(r \bowtie_{2=1} \pi_1(\pi_2(r) - s))$ holds, where $-$ denotes the difference of two sets. In order to ensure the applicability of this transformation, conjunctive queries need to be *safe*, i.e. every variable in a negated literal of the body must also occur in a non-negated literal. The representation uses a directed box-arc with the label "-". This can equally be applied to disjunction. However, disjunction is not supported by KAMEL.

Definition 3.1 describes a close relationship between the box representation and the relational algebra. In fact, the box representation is just another syntax for formulas in normal form with explicit equations and without constants. But using the box syntax, the translation procedure (described in the next section) turns out to become more easily understandable due to the graphical form and the distinction between non-terminals and attribute in the sense of attribute grammars. As a second advantage, the box syntax delivered directly the classes for an object-oriented implementation of the framework library (described in section 5).

4 Translation procedure

4.1 Basics

We use the compiler technique of attribute grammars [13, 4] to perform the translation of the box representation of a query into an equivalent query in the native language of an information source. If a grammar of the source language is given, translating a sentence (here: query in box representation) is done via backwards application of the production rules and evaluating semantic actions attached to each applied rule.

The grammar in figure 1 with a semantic action next to every rule is not complete but sufficient for the example of translating query 1 into its SQL-counterpart. The full grammar will be given in figure 2. Here, the A-boxes are used the first time. They represent the intermediate and final SQL-queries.

A resulting SQL-statement will consist of three lists: the from-, select- and where-lists. The first rule converts each N-box into an A-box. This is the *instantiation* step. The next two build up the query, dependent on the premises of the rules. They merge two SQL-queries and expand the where-list. In each step the select-list is rebuilt due to the remaining docks. The last rule converts an A-box, containing a query, into an M-box, containing a set of tuples. This is the *materialization* step, i.e. the query is sent to the information source and the answer is stored as an at-

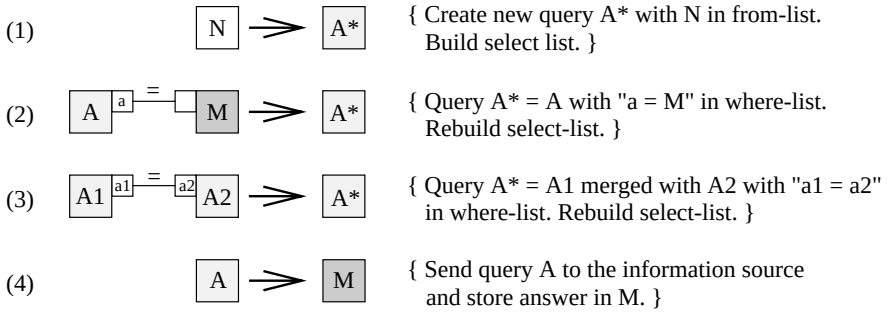


Figure 1: Rules and actions for SQL translation

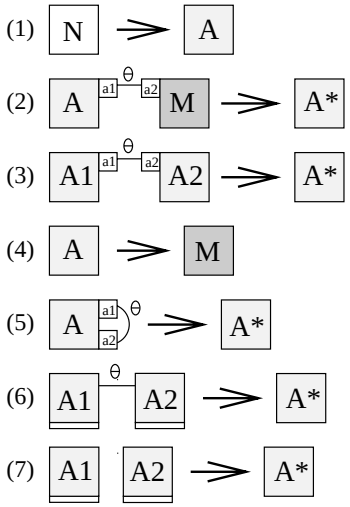
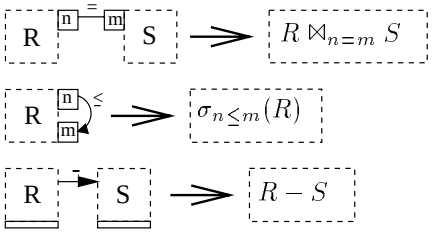


Figure 2: Grammar of the box representation

tribute in the resulting M-box. See the figure 3 of how the example translation takes place. Note that the number of docks is constantly minimized, so that only "useful" docks with connected arcs remain. This translation procedure is linear in the size of the query, i.e. in the number of boxes plus arcs.

Figure 2 shows all production rules of the box representation. As an additional rule, each low-level C-box firstly needs to be translated into an A-box using the same rules. If finally an M-box remains, the translation procedure is finished and returns its attribute as the result of the query.

In order to prove the correctness of a translator one has to reduce the result of a semantic action (that is a new A-box resp. query in most cases) to a relational expression like it is done in the definition of the semantics of the box representation. For example, given the reductions $red(A)$ and $red(M)$ in rule (2), the result $red(A^*)$ has to be $red(A) \bowtie_{n=m} red(M)$. The following picture shows three rules with their semantics:



Unfortunately, in most cases a formal definition of the native query language is not given, especially no definition using the relational operators. Therefore it is far from easy to prove the correctness of a query translator.

The examples given so far exemplified how an SQL-translator is build. Translating into SQL is an easy task due to the similarity in processing capabilities and data representation to KAMEL. The next subsection explains the idea of disabling rule actions according to the capabilities of an information source. Section 4.3 adds binding patterns to distinguish between output-docks and input-docks.

4.2 Disabling rule actions

If the question

$$\leftarrow equal\_closing('SAP', Name) : [\{today\}]$$

is posed to the mediator, the query

$$ans(N) \leftarrow stock(`SAP`, C) \ \& \ stock(N, C) \qquad (2)$$

must be answered by the ticker. The problems arise that firstly the ticker cannot process conjunctions (or joins) and secondly that the ticker does not accept constants ($`SAP`$) as arguments, since it only provides a stream of value pairs. Similar problems arise for example with inquiry systems in libraries. Such systems usually can not process conjunctions but do allow constants. Furthermore it is usually not possible to retrieve information about books for which the name of

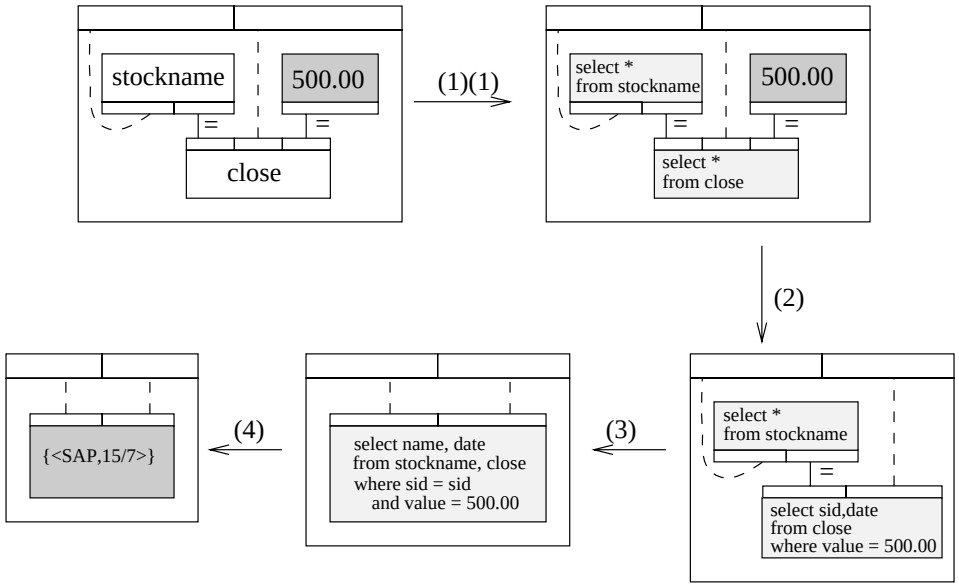


Figure 3: Translation of query 1

the author equals to the name of the title (for example with autobiographies). The same observations apply to fill-in-forms in HTML-pages in the world wide web.

To support those information sources which do not provide the necessary processing capabilities to answer conjunctive queries in general, the set of rules – or the grammar – is extended with the rules in figure 4.

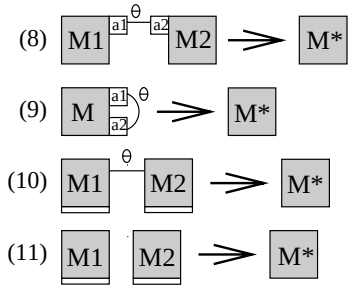


Figure 4: Additional production rules

These rules are only defined on M-boxes. Recall that M-boxes contain materialized relations as an attribute. If there is a unique interface to access the values in these relations, it is possible to implement the correct semantic actions for these new rules *independent* of the information source. In KOMET this interface is based on cursors or iterators. It is possible to read the value of a cursor, set the cursor to the first or the next value and test whether a next value exists. This is the usual cursor mechanism known from embedded SQL. Using cursors, there is no necessity to really materialize the relations.

As an example, the action of the rule (9) is implemented as follows: Every time the cursor of M^* is set to the next value, the cursor of M is carried on until the two values at arguments $a1$ and $a2$ are in θ-relation, e.g. equal. If no such position can be found, it is not possible to set the M^*-cursor one step further.

Using these additional rules and their source-independent actions, it becomes possible to choose between several sequences of rule applications. Figure 5 shows some paths for a query after invoking rule (1) two times. It is important to notice that it is always possible to find a path between the initial query in box representation and a single M-box containing the answer of the informations source, provided that the actions (1) and (4) are implemented. Therefore it is sufficient for a translator to enable only these two rules and disable the other ones. For information sources which do not have complete processing power it is indeed not *possible* to implement certain rule actions. For example, rule action (5) cannot be implemented for library inquiry systems. Of course the additional M-box-rules are always enabled. For SQL sources, all actions are implementable.

There are two indeterminisms in the translation procedure. Given a state of translation,

1. more than one rule may be applicable,
2. a rule may be applicable at different places.

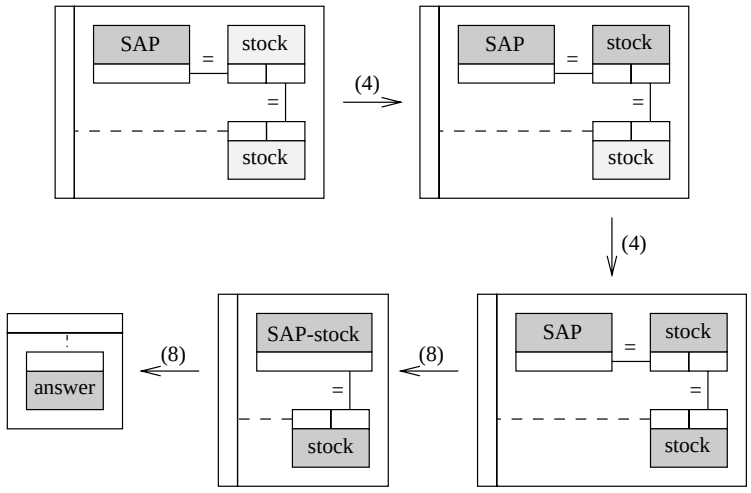


Figure 6: Translation of query 2

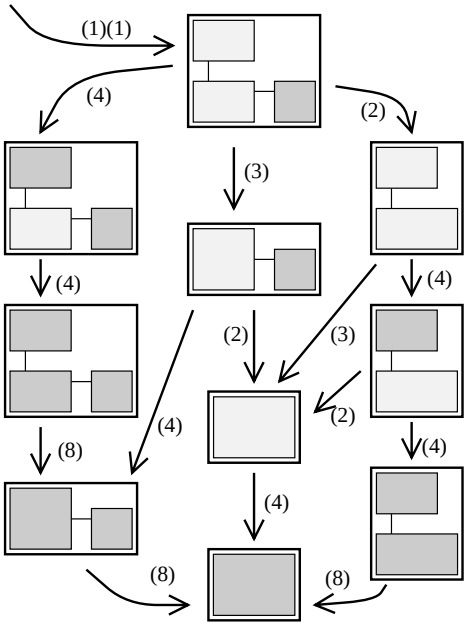


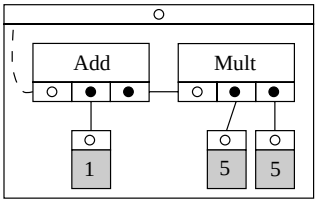
Figure 5: Some alternative paths of translation

Cost estimation, dependent on the size of relations and the speed of accessing the information source, could give preferences for one path over the others. For example, rule (4) should be avoided and only applied for small predicate extensions, if there is a choice.

The translation of query 2 for the ticker, after invoking action (1) two times, is performed as shown in figure 6. Only the rules (4) and (8) are needed. Rules (2) and (3) are not applied, because they are not implementable for such a weak information source. In both "stock"-M-boxes, the whole *stock*-relation is accessible via cursor-handles. The first application of rule (8) searches for the "SAP"-stock and the second for a stock with the same closing price.

4.3 Binding patterns

The usefulness of binding patterns for query translation has been emphasized in [19]. It is possible to integrate binding patterns into our framework. Each predicate in the export schema has a set of allowed patterns. A pattern determines for each argument of a predicate whether it is an input argument (i.e. this argument needs to be bound to a value), or an output argument (i.e. this argument gives a value as a result and has to be free), or both. Consider as an example a pocket calculator with the ability to add or multiply numbers. $Add()$ needs one output (the first) and two input (the second and third) arguments. Therefore $ans(X) \leftarrow Add(X, 1, 2)$, which stands for $X = 1 + 2$, is an allowed query, but $ans(X) \leftarrow Add(3, 1, X)$ is not. In the box representation, bound arguments are represented as $\bullet$-docks and free arguments as $\circ$-docks. For example the query $ans(X) \leftarrow Add(X, 1, Y)$ & $Mult(Y, 5, 5)$ is transformed into:



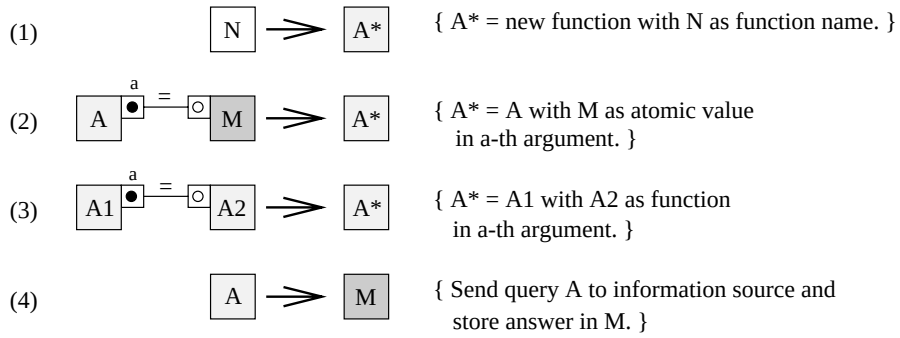


Figure 7: Rules and actions for functional translation

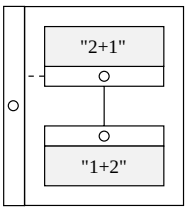
A query is *allowed* if each arc connects a •-dock with a ∘-dock and if every •-dock has exactly one arc. This ensures that every input-dock gets its required value from an output-dock. The example query meets this condition.

The rules with their actions in figure 7 specify the translator for a pocket calculator or for any source which processes concatenations of functions. A functional query has the general form of a function name with a set of arguments, which themselves can be functions or atomic values. All remaining rule actions are not implementable.

It is possible that the mediator poses a query to the translator which is not allowed. There are two possible binding conflicts: An arc connects two ∘-docks (∘-∘-conflict) or an arc connects two •-docks (•-•-conflict). A non-connected •-dock is a special case of the latter. As an example for an ∘-∘-conflict consider the following query:

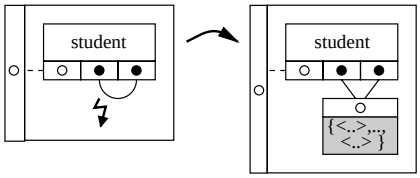
$$ans() \quad \leftarrow \quad Plus(X,1,2)\ \&\ Plus(X,2,1) \qquad (3)$$

which asks whether $1+2=2+1$. Note that no value is exported, so the answer will be Yes (= relation $\{\langle\rangle\}$) or No (= empty relation $\{\}$). After applying rule (2) four times, the box representation is the following:



Although applicable rule (3) is implemented, in this case the rule is *disabled* by force due to the ∘-∘-conflict. The translation goes on with rule (4) two times, which materializes the A-boxes, yielding the answer $\{\langle 3\rangle\}$ in both cases. Now the rule (8) is applicable with result $\{\langle\rangle\}$, i.e. "Yes". So: ∘-∘-conflicts are handled by the translator automatically via splitting the conjunction without consulting the mediator. It does so by means of disabling rules with conflicting arcs.

•-•-conflicts are resolved using *sort predicates*. Sort predicates evaluate to true for every instance of a sort, for example $natural\_number(X)$ is true for $X \in \{0,1,2,\ldots\}$. Consider an information source predicate $student()$ which returns the register number to any given first name and surname of a student. A query with a •-•-conflict is for example $ans(R) \leftarrow student(R,N,N)$, which requests all register numbers of students whose first name equals to the second name. The conflict is resolved by adding a sort predicate $names()$ which returns all possible names of students: $ans(R) \leftarrow student(R,N,N)\ \&\ names(N)$. In the box representation this is like cutting the •-•-arc and putting in the sort-M-box with an ∘-dock:

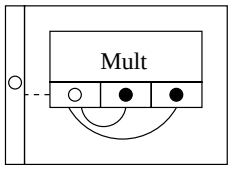


If the information source itself provides this sort predicate, the query is translatable as usual. If not, the query is either impossible to answer or the sort predicate has to be inserted by the translator. This may be a very time expensive conflict resolution – as in our example – and in cases where the sort predicate extension is not finite ($natural\_number()$), it is indeed impossible. But for small sorts (like Boolean values), this technique turns out to be useful.

For both conflicts the translator hides the impossibilities of processing as far as possible and divides or extends the query so that the demanded answer is given to the mediator. Together with the technique of

disabling rules, we have a powerful approach to specify query translators for various information sources. Both techniques complete each other. Reconsider for example the pocket calculator, which allows only limited binding patterns *and* does not support rules (5) to (7). Both techniques also overlap each other: For example the ticker problem encountered in section 4.2 can be solved either by disabling all rules but (1) and (4) or by allowing only ∘-docks.

There are still queries that cannot be handled. Examine the query $ans(X) \leftarrow Mult(X, X, X)$:



The box representation contains no conflicts. Since rule (5) is disabled for calculators, the N-box needs to be materialized using the rules (1) and (4). This is impossible because the box contains •-docks. Of course, rule (4) is only applicable if the A-box merely contains ∘-docks. This corresponds with the fact that a functional source cannot solve those fixpoint problems.

5 How to implement a query translator

In this section we shortly present the typical way to develop a translator using our approach. In a first step the developer of a query translator for a new information source has to determine how the knowledge of the source is accessed by the mediator, i.e. which predicates (with allowed binding patterns) make up the export schema. For example, the binary operations of a calculator (add,mult,etc.) are represented as 3-ary predicates, like $Add(\circ, \bullet, \bullet)$. Relational databases are usually represented as they are, because KAMEL is basically also relational.

In the second step s/he has to implement the structure of a query for the source – for example an SQL-statement consists of the select-, from- and where-lists –, the supported rule actions of the source (like shown in figure 1), and the cursor methods, which are dependent on the specific information source. To implement a full working translator, at least the actions for rules (1) and (4) have to be implemented. To put the most processing work into the source, all *possible* actions should be implemented. In KOMET this implementation is done in C++. A pre-developed translator-library of classes and program code defines the interfaces and many central methods like the translation loop, the action of the rules (8) to (11) and the communication with the mediator.

In many cases it is sensible to construct translators in a modular fashion. For example, a *pure* SQL-translator can define the structure of a query and all rule actions, except for rule (4). A *special* translator for Oracle as well as Informix can be build by *reusing* the pure SQL-translator and adding just the action of rule (4) (i.e. how to send a query to the source) and the cursor methods, which are also product specific. If two databases with different export schemas but the same DBMS are to be integrated in a mediator environment, the whole translator code for this DBMS can be reused.

Most of the programming work lies in the implementation of the source specific cursor methods, which include type conversion and native calling conventions. Much time can be saved, if industry standards (like ODBC for relational databases) are addressed, since most of the code can be reused for other standard sources. If the developer has sufficient knowledge of the product-specific programming interface, the amount of time for implementation of a translator can be counted in man-days.

With this framework we built translators for Oracle-7 databases, for WWW pages or files with a tabular structure and for Mathematica. Mathematica is a computer algebra system and processes – just like the pocket calculator – concatenations of functions. The corresponding translator was derived from a general translator for functional queries. In [11] a solution of how to address object-oriented databases has also been presented.

6 Conclusion and related work

In this paper we presented a new approach to query translation in a mediator environment. It is based on well known compiler techniques. By extending the grammar of the query language with production rules for which semantic actions can be defined once for *every* information source, the specification of translators for weak information sources turned out to be goal directed and easy. It is goal directed in the sense that the developer just ignores actions which cannot be served by the information source. The answering of queries which require more processing capabilities than the source provides is done automatically. Binding patterns have been added to the framework in a natural manner.

Many projects handle the integration of heterogeneous information sources in a mediator-like architecture, but only few address the problem of how to write

query translators. HERMES [2, 21] also uses a mediator language based on annotated logic. In contrast to KAMEL, the constraint part consists only of functions which the sources can process. So there is no difficulty in translating queries and all queries are guaranteed to be supported. This encoding hinders in building general conjunctive queries and combining two queries stated to the same source, as exemplified in query (5). A more general framework, which also includes binding patterns, has been presented in [3]. This work is comparable to the solution taken for the project Tsimmis [17] in [18, 19]. It is based on an enumeration of views which can be served by an information source. A query is rewritten in terms of these views. The views can be parameterized to express in one go a (maybe infinite) set of views. They are attached with semantic actions like done in our approach. The main difference is that [18] starts with *specialized* (but parameterized) views, i.e. it is necessary to express *each* supported query. In our approach in the first place it is assumed that a source supports *every* query, i.e. it starts with *general* views. It seems that the Tsimmis approach is best suited to support arbitrary information sources with the strangest (un-)abilities, while our approach focuses on the support of the most common weaknesses. Because of this focus, the development environment for translator specification in KOMET provides more specific support and the algorithms for translation are more efficient. None of the approaches mentioned support negation.

[9] presents a query representation comparable to the box representation but not in all its details. [10] firstly pointed out that compiler techniques are applicable for query translation.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addision-Wesley, 1995.

[2] Sibel Adalı and Ross Emery. An uniform framework for integrating knowledge in heterogenous knowledge systems. In *11th IEEE International Conference on Data Engineering*, pages 513–521, Taipei, Taiwan, March 1995.

[3] Sibel Adalı and Xiaolei Qian. Query transformation in heterogeneous information systems. `http://www.cs.umd.edu/projects/hermes`, October 1995.

[4] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers*. Addison-Wesley, Reading, Massachusetts, 1986.

[5] J. Calmet, D. Debertin, S. Jekutsch, and J. Schü. An executable graphical representation of mediatory information systems. In *12th IEEE International Conference on Data Engineering*, pages 124–131, New Orleans, March 1996.

[6] J. Calmet, S. Jekutsch, P. Kullmann, J. Schü, J. Svec, M. Taneda, S. Trcek, and J. Weißkopf. KOMET – Karlsruhe Open MEdiator Technology. Technical report, Institute for Algorithms and Cognitive Systems, University of Karlsruhe, 1996. `http://iaks-www.ira.uka.de/iaks-calmet/research/kamel-komet/`.

[7] J. Calmet and J. Schü. Design principles for secure integration of information systems. In G. Chroust and P. Doucek, editors, *Interdisciplinary Information Management Talks*, 1995.

[8] M.J. Carey et al. Towards heterogeneous multimedia information systems: The Garlic approach. Technical report, IBM Almaden Research Center, 1994. `http://www-i.almaden.ibm.com/cs/showtell/reports/database/Garlic_overview.ps`.

[9] B. Czejdo, M. Rusinkiewicz, and D. W. Embley. An approach to schema integration and query formulation in federated database systems. In *Proc. 3rd IEEE International Conference on Data Engineering, Los Angeles*, February 1987.

[10] D. I. Howells, N. J. Fiddian, and W. A. Gray. A source-to-source meta-translation system for relational query languages. In *Proceedings 13th VLDB Conference*, pages 227–234, Brighton, 1987.

[11] Sebastian Jekutsch. Design and implementation of a generic query translator for integration of heterogeneous information systems. Master's thesis, Institute for Algorithms and Cognitive Systems, University of Karlsruhe, June 1996. (in german; send email to jekutsch@ira.uka.de to get an english version).

[12] Michael Kifer and V.S. Subrahmanian. Theory of generalized annotated logic. *Journal of Logic Programming*, 12:335–367, 1992.

[13] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, June 1968.

[14] P. Kullmann, J.J. Lu, and J. Schü. Deductive query processing in mediatory hybrid knowledge bases with negation. In preparation, 1996.

[15] A.Y. Levy, D. Srivastava, and Th. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5:121–143, 1995.

[16] Jim Lu, Anil Nerode, and V.S. Subrahmanian. Towards a theory of hybrid knowledge bases. *To appear in IEEE Transactions on Knowledge and Data Engineering*, 1996.

[17] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In P. S. Yu and A. L. P. Chen, editors, *Proceedings of the 11th International Conference on Data Engineering*, pages 251–260. IEEE Computer Society Press, March 1995.

[18] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. D. Ullman. A query translation scheme for rapid implementation of wrappers. In *International Conference on Deductive and Object-Oriented Databases*, 1995.

[19] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *14th ACM Symposium on Principles of Database Systems*, pages 105–112, May 1995.

[20] V.S. Subrahmanian. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.

[21] V.S. Subrahmanian et al. HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995. `http://www.cs.umd.edu/projects/hermes/overview/paper`.

[22] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.