KOMET – A System for the Integration of Heterogeneous Information Sources

J. Calmet, S. Jekutsch, P. Kullmann, J. Schü

Institut für Algorithmen und Kognitive Systeme (IAKS)
Fakultät für Informatik, Universität Karlsruhe
Am Fasanengarten 5, 76131 Karlsruhe, Germany
{calmet,jekutsch,kullmann,schue}@ira.uka.de

Abstract. We present KOMET, an architecture for the intelligent integration of heterogeneous information sources. It is based on the idea of a mediator, which is an independent software layer between an application and various knowledge sources which need to be accessed. We present an especially suitable logic-based language for encoding typical mediation tasks like conditional preference strategies, schema integration or data inconsistency resolution. Using annotated logic, KOMET is able to perform various common types of reasoning, such as probabilistic, fuzzy, paraconsistent and certain types of temporal and spatial reasoning. In combination with an extensible type system and the embedding of external knowledge sources as constraint domains, our mediation language offers a rich framework, which not only facilitates access to structured information, but as well supports unstructured and semi-structured information. A number of examples show the practical application of our approach.

Keywords: Intelligent Information Systems, Knowledge Integration, Mediator

1 Introduction

The evolution of technology in the last couple of decades has led to a vast increase of information that is easily accessible with computers. But using all this information in connection is nearly impossible due to this very process yielding a broad variety of hardware platforms, operating systems, data management systems and data formats. Many information retrieval tasks are based on a multitude of information which is highly scattered among different information systems and services. To do a forecast on stock quotes one needs data on current and previous quotes, both normally available on completely different information systems.

The idea of such a so-called mediator system was first introduced by Wiederhold [Wie92]. His architecture makes use of a central component which facilitates the access to different knowledge sources without impairing the autonomy and heterogeneity of the involved sources. This approach has much in common with the federated database approach. But in contrast to federated database systems, the mediator approach uses a broader concept of *information source*. Not only traditional databases with different data models are considered as sources, but also unstructured information like world wide web pages, software packages like computer algebra systems, spreadsheet calculators or pattern recognition software and knowledge-based sources like expert systems or neural nets. Information sources are coupled to the mediator using *wrapper* or *translator* modules which map queries in the mediator format to queries in the source- specific query language (Figure 1). Another major task of these components is the transformation of the retrieved data from the source-specific representation into the mediator data model.

In this paper we present KOMET, a shell for developing dedicated mediators by means of a declarative language. There are few projects which deal with information integration in a comparable generality. Closely related projects in this field of research are HERMES [AE95] and TSIMMIS [PGMU96]. HERMES also uses annotated logic for the mediatory knowledge base, but uses a different approach for the query translation. Its rule-rewriting approach requires much more a priori anticipation of the kind of queries that might occur. Only queries for which corresponding templates have been established can be send to a data source. The language of HERMES does not support negation and is not as flexible in its type system. To the best of our knowledge, KOMET is the first system that implements the evaluation of the well-founded semantics for annotated logic programs. TSIMMIS emphasizes the integration of semi-structured information sources. It makes use of a special data model for representing this information. In our opinion, this approach has the major drawback that even though structured information can be represented, its processing is far less efficient than would be possible by retaining the structure. We have deliberately chosen a structured approach, since it adequately supports structured data sources in terms of efficiency. At the same time our framework facilitates the integration of unstructured and semi-structured information due to its extensible type system.

In the sequel, first the architecture and language of the KOMET system are described. The third section gives some examples that show the practical use of the KOMET language. Section four describes how unstructured and semi-structured information can be represented and processed in our framework. Finally, the conclusions section discusses future directions of our work.

2 KOMET – A Mediator System Based on Annotated Logic

The KOMET (**K**arlsruhe **O**pen **ME**diator **T**echnology) system[1] is a knowledge-based mediator shell for intelligent information integration. From the user's perspective, the system provides a set of views on the underlying data. On the basis of these views, the user can pose queries to the system in a uniform way without having to consider the location and format of the underlying knowledge. The basic KOMET architecture is depicted in figure 1.

[1] `http://iaks-www.ira.uka.de/iaks-calmet/research/kamel-komet/`

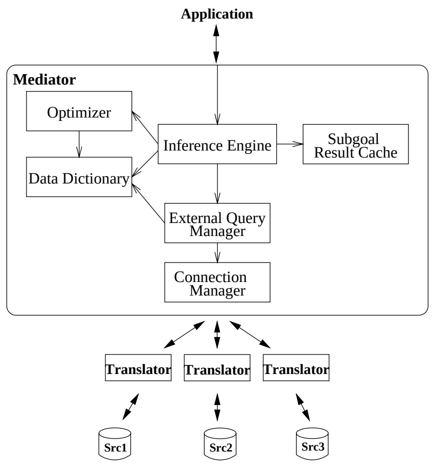

Fig. 1. Mediator Architecture.

The heart of our system is an efficient engine for evaluating annotated logic programs under the well-founded semantics, enhanced with constraints for integrating external knowledge sources. The engine uses SLG resolution [CW93] which has following advantages over other evaluation procedures:

– SLG resolution is a partial deduction procedure, consisting of several fundamental transformations. A query is transformed step by step into a set of answers. The use of transformations seperates logical issues of query evaluation from procedural ones.
– SLG resolution is sound and complete with respect to the well-founded partial model for all non-floundering queries.
– The computation rule for selecting a literal from a rule body as well as the control strategy for selecting transformations to apply are arbitrary.
– It avoids both positive and negative loops and always terminates for programs with the bounded-term-size property [CW93].
– For function-free programs, it has polynomial time data complexity for well-founded negation.

Similar to OLDT resolution[2], tabulation is used to avoid redundant computations. It is additionally needed to guarantee termination. We have extended SLG resolution for processing of annotated logic programs with a many-sorted type system.

Besides the inference engine, the KOMET system core comprises the *Subgoal Result Cache*, the *External Query Manager* with a *Connection Manager*, the *Optimizer* and the *Data Dictionary*. The *Subgoal Result Cache* is used for tabulation during the resolution process. A buffering strategy keeps query results

[2] **O**rdered **L**inear resolution for **D**efinite clauses with **T**abulation

across queries if suitable. The *External Query Manager* appears as a constraint solver in the mediator. It decomposes constraint parts of clauses with the help of the *Optimizer* and sends partial conjunctive constraints to the *Translators* which correspond to individual information sources. The *Connection Manager* coordinates and monitors network access. The *Data Dictionary* contains meta data about all registered information sources, relations, functions and data types. Systems predicates allow the access of this information and thus facilitate meta reasoning.

The integration of arbitrary information sources into a mediator system requires transformations on different levels. On the semantic level the source-specific schema needs to be transformed into a schema useful for the mediation task. In our framework, this is done by the inference engine using schema integration rules. On the data representation level information needs to be transformed from the source-specific data model into the common data model and vice versa. Figure 2 illustrates the levels of integration.

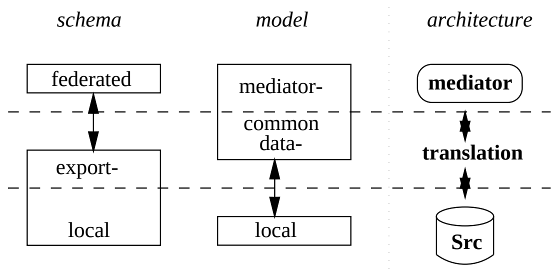

Fig. 2. Translation of model and schema

In our architecture translation connotes transformation of queries posed by the inference engine into the source-specific query language (e.g. SQL) as well as the involved data model transformations. We have developed an environment for rapid implementation of query translators for arbitrary types of information sources [CJS97]. Based on compiler techniques, it uses rules which describe how the constructs of the mediator language are mapped to the native language constructs of a specific source. This approach can be used to build translator hierarchies and allows the reuse of certain translator functionality. In the remainder of this paper, we will focus on how the KOMET language can be used to handle typical mediation tasks.

3 The KOMET Language

The KOMET language is a declarative language which implements a many-sorted annotated logic [KS92] with a high degree of flexibility. It provides a set of basic data types and truth value sets which can be extended using a programming interface. The KOMET language defines the common data model.

In annotated logic, each fact has a truth value appended to it. Annotated logic only imposes some restrictions on the algebraic structure wihich the set of truth values has to form. This property allows to choose among a number of common logics, which range from two-valued logic to fuzzy logic and certain types of temporal and spatial logic. In this sense, annotated logic is generic.

A KOMET program is made up of a set of constant symbols, variable symbols, function symbols and predicate symbols. Since KOMET is based on annotated logic, each predicate declaration requires the assignment of a truth value type.

STRING and FOUR are predefined data and truth value types, respectively. FOUR comprises `t` (true), `f` (false), `u` (unknown) and `c` (contradiction). For simplicity we will use this truth value type for the following examples. Example 1 shows how recursive clauses allow us to calculate the *transitive closure* of a predicate which is not possible with plain SQL. Due to the use of well- founded semantics, recursive clauses which contain negation are also allowed. For the program in the following example, the query `isAncestor(X,Y):[t]` would return the three results `(X=Paul,Y=Ann)`,`(X=Ann,Y=Mary)` and `(X=Paul,Y=Mary)`.

Example 1. Clauses

```
#predicates
isChild(STRING,STRING):[FOUR]
isAncestor(STRING,STRING):[FOUR]

#clauses
isAncestor(X,Y):[t] <- isChild(Y,X):[t]
isAncestor(X,Y):[t] <- isAncestor(X,Z):[t] & isChild(Y,Z):[t]
isChild('Ann','Paul'):[t]
isChild('Mary','Ann'):[t]
```

To be able to limit the data which defines a derived predicate, KOMET allows the use of constraints which can be constructed form constraint relations and functions. In our system, constraint relations have the general form *domain::function(argumentlist)* and can be used in the conjunction of a clause body. Naturally, each data type such as ULONG provides a number of predefined domain functions and relations for the basic boolean and mathematical operators. Each data type and each truth value type therefore represents a constraint domain in the KOMET system.

The KOMET system looks at an *external knowledge source* also as a constraint domain which supplies a set of functions and relations which represent the knowledge that is available from the knowledge source. Following example shows how databases can be introduced.

Example 2. Mediator program

```
#domains
DB1 = ODBC(MyDb.CFG)
```

```
DB2 = DDBC(HisDb.CFG)

#predicates
Address1(STRING,STRING,STRING):[REAL01]
Address2(STRING,STRING,STRING):[FOUR]
Names_DB1(STRING,STRING):[FOUR]

#clauses
Address1(X,Y,Z):[1.0] <- DB1::ADDRESS(X,Y,Z)
Address1(X,Y,Z):[0.5] <- DB2::ADDRESS(X,Y,Z)

Names_DB1(X,Y):[t] <- DB1::ADDRESS(X,Y,Z)
Address2(X,Y,Z):[t] <- DB1::ADDRESS(X,Y,Z)
Address2(X,Y,Z):[t] <- DB2::ADDRESS(X,Y,Z) &
                       not Names_DB1(X,Y):[t]
```

`MyDb.CFG` refers to a configuration file which lists the available functions and predicates. The information in this file is called *export schema* (see figure 2) of an information source, expressed in the common data model of KOMET. Dependent on the knowledge source, this file may also contain information on how these functions are mapped onto the functionality of the knowledge source.

Example 2 shows how to build a mediator for two telephone databases which contain conflicting data. DB1 is assumed to be reliable whereas we are not confident in DB2. The mediator program shows two possibilities for resolving possible conflicts. The definition of `Address1` does not suppress any data, it just appends a truth value to each data set, depending on its origin. The predicate `Address2` will prefer the telephone number from DB1 if there is an entry in both databases for one person. This way, incorrect entries will not be visible at all.

4 Mediation Examples Using the KOMET Approach

In the following we will show how typical mediation tasks can be expressed in the KOMET framework using clauses and annotations. For the sake of a shorter and clearer presentation, we use a slightly simplified syntax and avoid obvious declarations in this section.

Knowledge Inconsistencies

In a mediatory environment, it may be necessary to distinct knowledge from different sources. It is useful to rename the predicates so that objects from different knowledge sources are disjoint. Any identities between predicates must then be reestablished explicitly. The annotation lattice $\mathcal{P}(\{Src_1, \ldots, Src_n\})$ with the usual subset ordering is introduced for this purpose, where Src_i is an identifier for an external domain.

In the following we provide two examples how annotated logic may help:

– Conflict solving: Using lattices like $\mathcal{FOUR}$[3] it is possible to represent clauses

[3] consisting of $\bot$ = *unknown*, t = *true*, f = *false* and $\top$ = *inconsistent*

which explicitly state how to deal with conflicting knowledge, e.g.

$$buy\_stock(IBM) : [\{m\}, f] \ \leftarrow \ buy\_stock(IBM) : [\{Src_1, Src_2\}, \top]$$

If the knowledge sources Src_1 and Src_2 disagree ($\top$) on whether to buy IBM, the mediator (m) rather should be careful about it.

– Preference of sources: The following clauses express a preference of a source with respect to a particular single proposition. It is a somewhat complicated case: Src_1's value is more reliable in the case that it differs by no more than 10 from the value of Src_2. In the other case, Src_2's value needs to be stored as the mediators opinion about the salary.

$$\begin{aligned} Salary(Name, Sal) : [\{Src_1\}, t] &\leftarrow DB1 :: Salary(Name, Sal) \\ Salary(Name, Sal) : [\{Src_2\}, t] &\leftarrow DB2 :: Salary(Name, Sal) \\ Salary(Name, Sal) : [\{m\}, t] &\leftarrow Abs(Sal - Sal2) \leq 10 \ \& \\ &\quad\ Salary(Name, Sal) : [\{Src_1\}, t] \ \& \\ &\quad\ Salary(Name, Sal2) : [\{Src_2\}, t] \\ Salary(Name, Sal2) : [\{m\}, t] &\leftarrow Abs(Sal - Sal2) > 10 \ \& \\ &\quad\ Salary(Name, Sal) : [\{Src_1\}, t] \ \& \\ &\quad\ Salary(Name, Sal2) : [\{Src_2\}, t] \end{aligned}$$

Semantic Similarities

A relation $Professor(Id, Name)$ in the domain DB1 and $Secretary(Id, Name)$ in the domain DB2 may be *generalized* in the context of the university administration office:

$$\begin{aligned} Staff(Id, Name) : [\{m\}, t] &\leftarrow DB1 :: Professor(Id, Name) \\ Staff(Id, Name) : [\{m\}, t] &\leftarrow DB2 :: Secretary(Id, Name) \end{aligned}$$

There is also the possibility to express semantic proximity by means of fuzzy values [SK93].

Domain Incompatibilities

– Synonyms: In two different knowledge sources the identity between attributes may be established by the following clauses:

$$\begin{aligned} Employee(Id) : [\{m\}, t] &\leftarrow DB1 :: Employee(Id, Name) \\ Employee(Id) : [\{m\}, t] &\leftarrow DB2 :: Personnel(Id, Name) \end{aligned}$$

– Data Representation Conflicts: Suppose that in the above example in the DB1 database the Id is defined as a 9 digit integer whereas the Id in DB2 may be defined as a String. Thus, data conversion is required by means of a constraint function $Convert\_Str\_to\_Int$ which is provided by the domain SYS:

$$\begin{aligned} Employee(Id) : [\{m\}, t] \leftarrow DB2 &:: Personnel(Name, SId) \ \& \\ Id &= SYS :: Convert\_Str\_to\_Int(SId). \end{aligned}$$

Schema Conflicts

- Schema Isomorphism Conflicts: In this conflict, semantically similar entities have a different number of attributes.

$$Instructor(No, Phone) : [t] \leftarrow DB1 :: Instructor(SS, Phone)$$
$$Instructor(No, Phone1) : [t] \leftarrow DB2 :: Instructor(SS, Phone1, Phone2)$$

- Missing Data Item Conflict: Analogously default values can be defined, or values might be calculated if they are missing in one source.

5 Unstructured and Semi-structured Information

One goal in the development of KOMET was to support different types of information sources in the most efficient manner. This can be achieved by exploiting a source's characteristics and abilities as far as possible. A structured approach has the advantage that it can efficiently support well-structured information which is the case for a large part of the available information. At the same time, our structured approach allows the support of unstructured and semi-structured information as it is flexible enough for representing and processing these types of information. The key feature for this capability is the extensible type system of KOMET.

5.1 Unstructured Information

A basic KOMET type can be regarded as an abstract data type for which functions and relations have to be defined according to a certain minimal programming interface. Additional functions and relations can be incorporated into the KOMET type and used in the constraint part of clauses. However, KOMET has no knowledge of the internal structure of a data type[4]. This property is the basis for incorporation of unstructured information. Consider a text archive where each text is stored in an individual file. Additionally, assume we have a text analysis software package which scans texts for a number of keywords. For use in KOMET, we would create a new data type `TEXT`, which can e.g. be constructed from an existing C++-class. For the implementation of this type we have the choice of storing the text *itself* in a data item or using some kind of handle like a file name. Next, we must construct a constraint domain which allows us to access the texts files of the archive. Finally, we can add a relation `CONTAINS` to this domain, which takes a list of keywords and a text as arguments and determines whether the text contains all of them or not.

[4] A special case are types which have been defined in a mediator program with the KOMET language

Example 3. Unstructured information

```
#sorts
KEYLIST = LIST(STRING)

#predicates
Search(TEXT,KEYLIST):[FOUR]

#clauses
Search(X,Y):[t] <- ARCH::FETCH(X) & ARCH::CONTAINS(Y,X)
```

The above example shows a mediator program that retrieves all texts that contain a list of keywords. Note, that the internal representation has no influence on the coding of the mediator program. It does influence the implementations of the involved relations though. It is up to the calling application to interpret data items of type `TEXT` appropriately.

5.2 Semi-structured Information

The KOMET type system can as well be used for representing semi-structured information. Together with some functions and relations for access, processing and aggregation we are able to process this kind of information in our framework in a similar manner as described in related projects [PGMU96]. The following simple example shows how this could be achieved. `FLEXTYPE` denotes a data type which can hold different data types and corresponds to the pair *(type,value)*.

Example 4. Semi-structured information

```
#sorts
OBJECT = STRUCT(OID:STRING,LABEL:STRING,DATA:FLEXTYPE)

#predicates
Address(OBJECT):[FOUR]

#clauses
Address(OBJECT::MERGE(A1,A2)):[t] <-
        SRC1::FETCH(A1) & SRC2::FETCH(A2)
        OBJECT::EQ(A1,'LastName',A2,'LastName')
```

The `MERGE` function merges two objects of type *set* into one new object of type *set* and assigns a new unique object identifier to it. Relation `EQ` takes two objects of type *set* and checks two of its elements which are referenced by their label for equality. The given program demonstrates how semi-structured objects from different sources may be merged in the mediator. Clearly, many other operations necessary for integration can be realized in our framework.

6 Conclusions

In this paper we presented the KOMET environment for building mediators. Its language and architecture is well suited to serve typical mediation tasks in a heterogeneous environment. The language has formal semantics and is declarative, which supports easy construction of the mediatory knowledge base. The extensible type system together with the felxible concept for accessing external knowledge also facilitates processing of unstructured and semi-structured information.

KOMET will be applied in the project STEM[5] which aims at developing a software system to assist land managers in environmentally sensitive areas with long term sustainable decisions.

The core functionality described in this paper has been realized in a prototypical system. It is coded in C++ and runs under the Solaris and Windows operating systems. A graphical front end for debugging mediator clauses [CDJS96] and for semi-automatic construction of schema integration clauses have also been realized. Theoretical work has been done on view maintenance in a mediator architecture [Sch95].

Future work will focus on different aspects of query optimization and security in mediator systems. On the knowledge level we will investigate knowledge acquisition and use of common ontologies in the mediator context.

References

[AE95] S. Adalı and R. Emery. A uniform framework for integrating knowledge in heterogenous knowledge systems. In *Proc. 11th IEEE International Conference on Data Engineering*, pages 513–521, Taipei, Taiwan, March 1995.

[CDJS96] J. Calmet, D. Debertin, S. Jekutsch, and J. Schü. An executable graphical representation of mediatory information systems. In *Proc. 12th IEEE International Conference on Data Engineering*, pages 124–131, New Orleans, March 1996.

[CJS97] J. Calmet, S. Jekutsch, and J. Schü. A generic query-translation framework for a mediator architecture. In *Proc. 13th International Conference on Data Engineering, Birmingham, U.K.*, pages 434–443, April 1997.

[CW93] W. Chen and D. S. Warren. Query evaluation under the well-founded semantics. In *Proceedings of the 12th Annual ACM Symposium on Principles of Database Systems*, pages 168–179. ACM, ACM Press, 1993.

[KS92] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming. *Journal of Logic Programming*, 12(1):335–367, 1992.

[PGMU96] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proc. 12th International Conference on Data Engineering*, pages 132–141. IEEE Computer Society, February 1996.

[Sch95] J. Schü. *Updates and Query-Processing in a Mediator Architecture*. PhD thesis, Universität Karlsruhe, 1995.

[SK93] A. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems*, pages 283–312. IFIP, Elsevier Science Pubslihers, 1993.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[5] `http://www.cogsci.ed.ac.uk/ stem/`