

Presented at 2nd International Workshop on User Interfaces to Data Intensive Systems, Zurich, Switzerland, May 31. - June 1. 2001

An Object-Oriented Multi-Mediator Browser

Kristofer Cassel and Tore Risch

Uppsala Database Laboratory
Dept. of Information Science
Uppsala University
Sweden

November 29, 2000

ABSTRACT

The area of data integration has gained increased popularity in recent years. This paper describes a data browser for a data integration system where an intermediate layer of a distributed multi-database system, a mediator database system, is used to query and integrate data from heterogeneous data sources. The data sources can be regular relational databases but also other data producing programs. They often have complex data representations and are often object-oriented (OO). The mediator database layer is therefore also object-oriented for a high abstraction level. An OO query interface is used to access the mediator layer from application programs and users. For a scalable and component based architecture the mediators can be used as servers for other mediators. This leads to a distributed mediator architecture where mediator servers interact with other mediators and data sources. An OO multi-mediator browser, GOOVI, is presented which enables maintenance of such distributed mediator databases. With GOOVI the autonomous mediators in a federation can be viewed, queried, and updated. This multi-mediator browser also provides user interfaces for integrating data through OO views. The paper describes the functionality and architecture of GOOVI.

1. Introduction

With the mediator/wrapper approach to data integration [Wie92] *wrappers* define interfaces to heterogeneous data sources while *mediators* are virtual database layers where queries and views to the mediated data can be defined. The mediator approach to data integration has gained a lot of interest in recent year [BE96,G97,HKWY97,LP97,TRV98]. Early mediator systems are central in that a single mediator database server integrates data from several wrapped data sources. In the work presented here, the integration of many sources is facilitated through a scalable mediator architecture where views are defined in terms of object-oriented (OO) views from other mediators and where different wrapped data sources can be plugged in. This allows for a component-based development of mediator modules, as early envisioned in [Wie92].

This paper describes GOOVI (Graphical Object-Oriented View Integrator), which is an OO graphical browser for Amos II [RJ00,RJK00,JR99a]. Amos II is a distributed OO mediator system that allows OO views in mediator servers to be defined in terms of views in other mediator servers on the net. Data sources are wrapped by embedding them in Amos II mediator servers through foreign data source interfaces [RJ00,JR99a]. GOOVI provides a graphical user interface to such a federation of Amos II mediator servers. The primary GOOVI user is a mediator administrator who defines and modifies distributed mediators and who investigate properties of them. Such a mediator administrator needs to find mediators in a federation, to inspect and modify the schema of each individual mediator, to inspect and update the contents of each mediator, and to define integrating views of data from other mediators. The actual wrapping of data sources requires some programming and is outside the scope of this paper.

For finding relevant mediators in a federation GOOVI provides interfaces to browse some general meta-mediator properties of the federation members, such as their names and locations.

To inspect the schema of each mediator, GOOVI provides a very easy way to graphically browse several AMOS II mediator databases in a federation. The type hierarchies of different mediators can be visualized in separate windows. GOOVI also has primitives for schema modification.

For manipulating the contents of the mediator databases, individual database object instances in each mediator database can be both inspected and modified. The interface illustrates on a high level the relationships between database objects. Furthermore. OO queries can be submitted to any mediator server in the federation. The results of such queries are presented as browser objects too, for convenient further browsing. Separate browser and query window groups can be opened for browsing different Amos II mediators in a federation.

For integration of data from different mediators GOOVI provides a graphical interface to define OO multi-mediator views using multi-database integration primitives of Amos II [JR99a, JR99b].

To make the user interface easy to use and more consistent with other similar tools we decided to make it appear in similar style as the user interface of the classical Windows Explorer. This is a compact and attractive design that follows standards set by many other GUIs that perform similar tasks, which will help the user to work fast and without problems and confusion.

This paper first discusses related work. Then the style of interactions with GOOVI is presented in Section 4. Section 5 describes how to define multi-mediator views, followed by an overview of the implementation in Section 6.

2. Related work

Most commercial relational databases provide graphical database browsers. A few systems address browsing of OO databases [MDT88, F89, AGS90, CHMW96, CA96]. As in Pesto [CHMW96] the results of queries in GOOVI are returned as database structures that can be examined by GOOVI as any other database objects. This is called *query-in-place* in Pesto. As in Jasmine [CA96] we use an interface style with a Windows oriented look-and-feel. However, unlike Jasmine (and like Pesto) our query interface is stream oriented to allow for retrievals of large sets of data.

An XMLQL-based browser for accessing a single mediator is proposed in [B99]. A major difference between GOOVI and all other known database browsers is that GOOVI is designed for browsing *federations* of OO databases, as needed in our distributed mediator environment. This puts new requirements on the browser to be able to separately visualize several OO databases in a federation of mediator servers and also to be able to graphically define views of data from several of these databases. GOOVI therefore allows the user to open separate OO *database browser windows* for each members in the federation. Separate windows, e.g. for querying and data integration, can be associated with each such database browser window. A *connection manager* allows the user to select among the members of the federation to open up new database browser windows. Furthermore, *data integration windows* can be defined where OO multi-mediator views can be defined that combine data from several mediator servers.

The interaction between GOOVI and the mediator in the federation respects the autonomy of the members. This means that, unlike other database browsers, GOOVI does not presuppose any central database, mediator, or global conceptual schema. The only requirements are that the databases in the federation are wrapped in Amos II mediators and that they are registered

with a meta-mediator called the *mediator name server*. The mediator name server is an ordinary mediator server having names, locations and some other *meta-mediator properties* of the members in a federation. However, the mediator name server contains only very limited information about the members of the federation; it respects the autonomy of the members and it is *not* a full central data dictionary.

3. Browsing Multi-mediators

A federation of Amos II servers may contain many mediator servers distributed on a computer network. Each mediator server is an autonomous database server supporting the OO query language AmosQL of Amos II having a syntax and semantics similar to the OO parts of SQL-99 [RJ00,RJK00]. The mediator servers have primitives for communicating with other mediator servers and for wrapping external data sources. Foreign data can be made accessible to the federation by developing an interface, called a *wrapper*, for each kind of data source where simple query algebra operations are executed on external data elements. We have successfully developed wrappers for ODBC, STEP/EXPRESS, XML, and web-based search engines.

If a data source has no own query processing capabilities, as e.g. STEP/EXPRESS and basic XML, the processing of the imported data is done inside Amos II in a streamed fashion or through materialization in the mediator database, depending on the data source.

However, if the data source accepts queries, as e.g. ODBC and search engines, there is also a need to develop query *translators* that translate AmosQL queries into execution plans containing query fragments submitted to the foreign query engine for evaluation.

From a mediator administrator's perspective the mediator layer appears as a federation of distributed and autonomous mediator servers each using the same OO data model and query language, where some mediators wrap one or several data sources while other combine data through OO views over data from other mediators.

Figure 1 illustrates one such scenario where we have five mediator servers named *WC*, *NAMESEVER*, *TORE*, *HOLIDAYFLIGHGTS_INC*, and *FLYGRESOR_AB*¹ running somewhere on the network. The mediator named *NAMESEVER* is a mediator server knowing the locations, names, and other properties of the mediator servers in the federation, i.e. the *mediator name server*.

¹ Air Travel Inc. in Swedish.

The other mediator servers will query the mediator name server using AmosQL when they need to know meta-properties of members in the federation. Notice here that this architecture does not include any global conceptual schema; every autonomous mediator server has its own local OO schema and the mediator name server knows only very general meta-properties of the other mediator servers.

In the example, the two mediator servers *WC* (World Cup) and *TORE* wrap and translate data from some data sources, e.g. relational databases. The top double box is the GOOVI system itself. When GOOVI is started it may first act as a stand-alone system connected to its own private local Amos II database called *LOCAL*. The user can browse and store private data in the local database. It is also used internally by GOOVI for caching data extracted from other databases. The user can ask GOOVI what mediator servers are available in the federation. GOOVI will send queries to the name server to find this out.

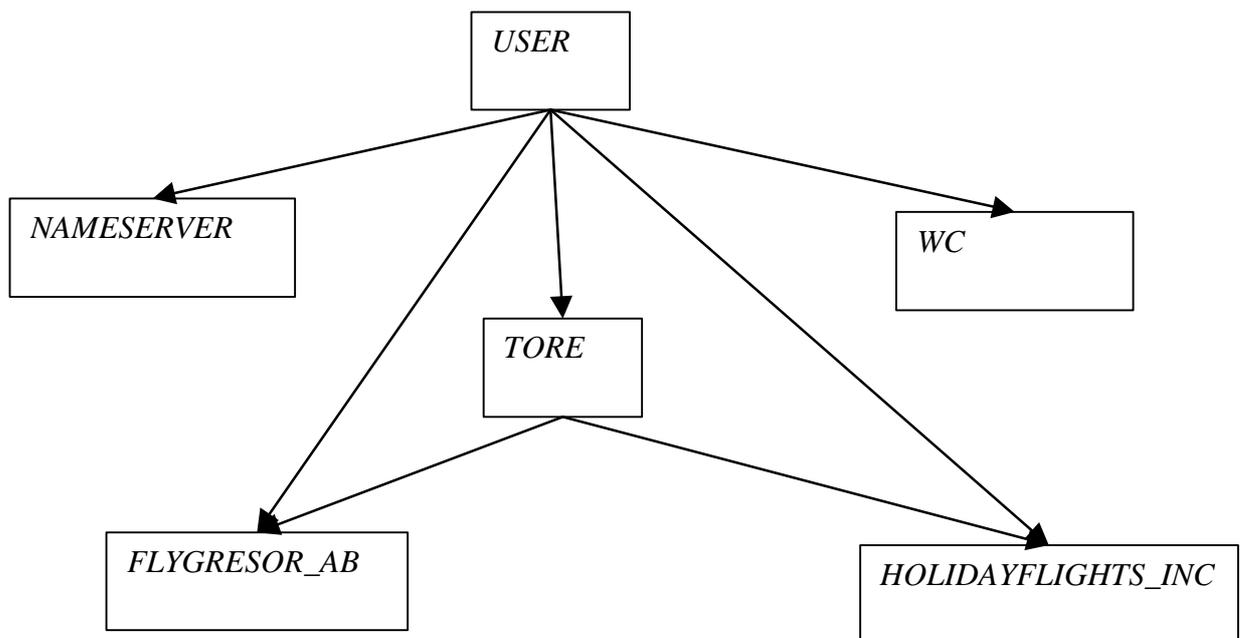


Figure 1: Multi-mediator Scenario

Figure 2 shows an example of a GOOVI interaction in our multi-mediator scenario. In the example two multi-mediator type browser windows are open, the top one connected to the *TORE* mediator and the bottom one to the mediator server named *WC* for which also a query is stated. The result of the query is presented as a scan of objects of type *PLAYER*. Furthermore,

Figure 2 also shows how the user can ask GOOVI what mediator servers are available in the federation by opening a *mediator browser* to the right. GOOVI has here internally queried the name server mediator for the names, locations, ports, etc. of the mediator servers in the federation. Through the mediator browser dialog the user can open separate type browser windows on any mediator server in the federation.

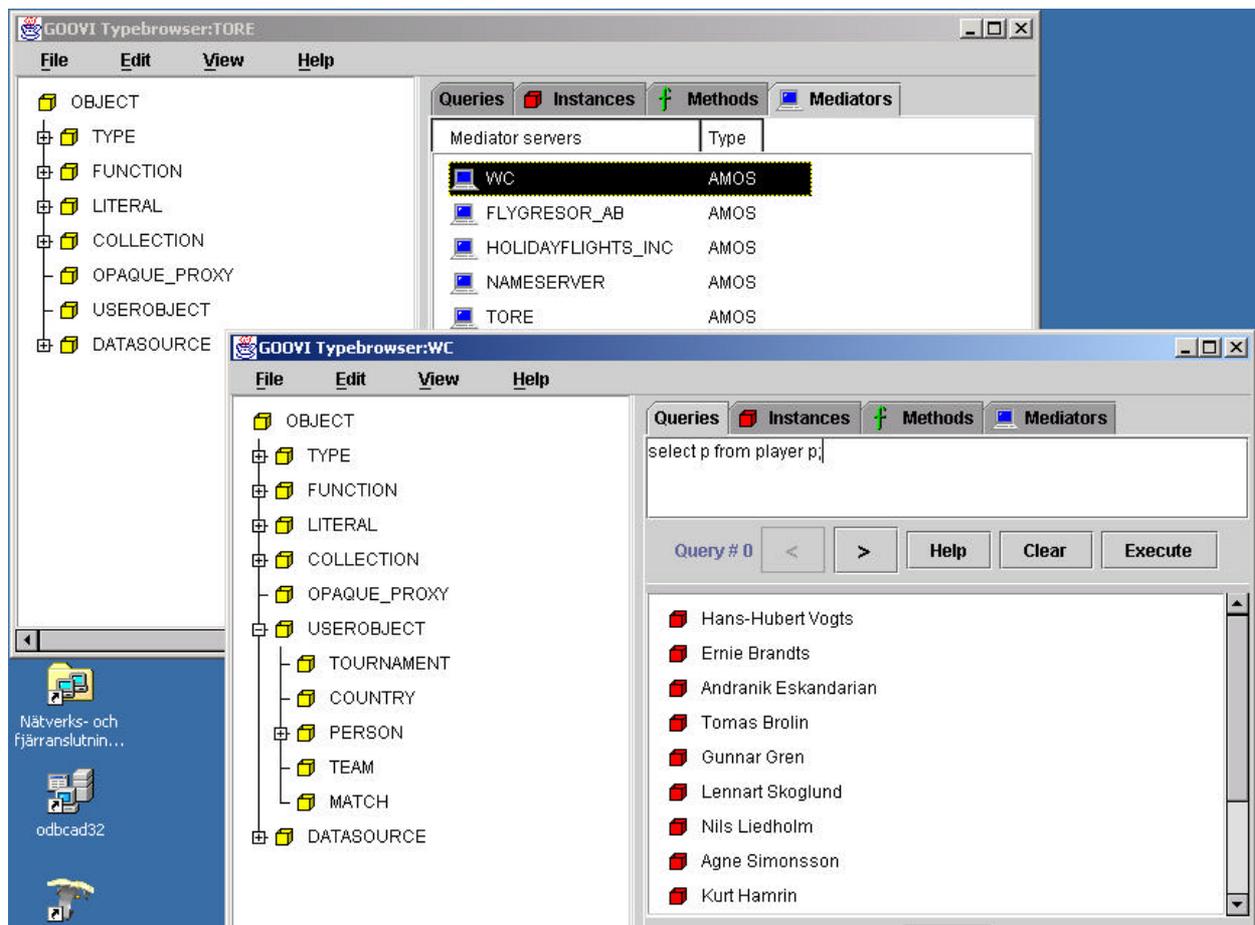


Figure 2: GOOVI Interaction

When the mediator browser is opened it has to contact the mediator name server to obtain the set of mediators in the federation. The mediator name server can be located anywhere on the net and the user is therefore first asked to specify its network location to GOOVI. The members of the federation can be located anywhere on the net too. Through queries to the mediator name server the mediator browser can present the user with a menu of the members of the federation.

3.1 The Type Browser

The *type browser window* graphically displays the type hierarchy for a particular mediator server. Initially the type browser window for the local database of GOOVI is displayed. The

approach of having the type hierarchy as the main view of the database is a natural choice since the types and their relationships are central for an OO database. This approach was also chosen for the object browsers of the OO DBMSs Jasmine [CA96] and Iris [F89].

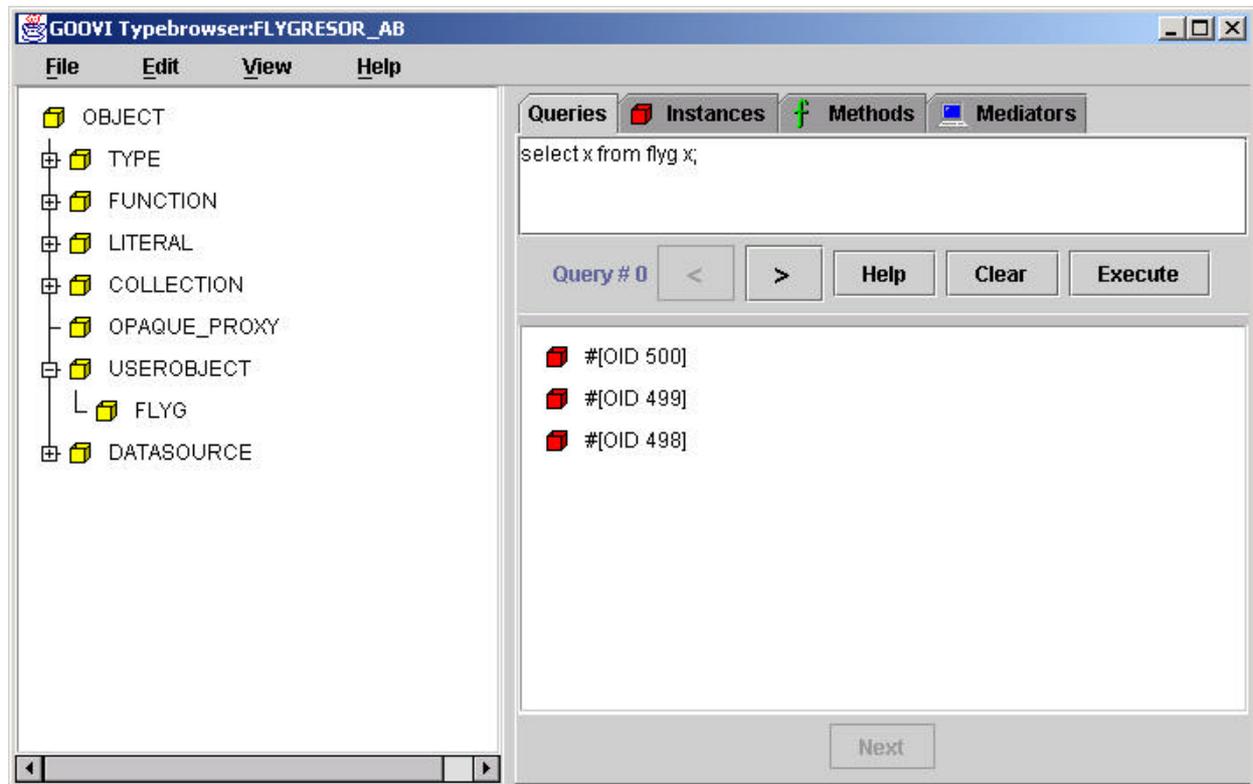


Figure 3 The GOOVI Type Browser

Figure 3 shows the type browser window for a mediator database *FLYGRESOR_AB*, wrapping a data source containing flight data for a Swedish travel agent. The window is divided vertically in two parts with a left panel showing the type hierarchy and a right panel having a number of tabs for inspecting different properties of the mediator. There is also a menu bar on top with functions for creating new types, viewing, searching, editing, etc. Some of the menu-items have keyboard shortcuts.

When opening browsers of several AMOS II mediator servers, separate and independent type browser windows appear on the screen for each mediator, as shown in Figure 2. Every type browser window thus has a one-to-one relation to a connection to an AMOS II database in the federation. This helps the user to keep track of what mediator databases are being inspected. By closing the corresponding type browser window the connection is also closed along with all other dialogs associated with the closed type browser. The type browser window thus serves as a grouping of all GOOVI dialogues associated with a particular mediator server. The name of the AMOS II database (e.g. *FLYGRESOR_AB*) is stated in the title of the type browser. The initial type browser window is *LOCAL* (the local database) from which the user can use the mediator browser to open connections to other mediators in the federation. When the last type browser

window is closed GOOVI is exited after a confirmation.

3.2 The Query Editor

The *query editor* allows execution of queries and browsing the resulting scans displayed in a graphical format. The entities of query results are inspectable and have a consistent appearance as other database objects for generality and user orientation. We have chosen to specify queries as text input to browser forms rather than using elaborate mouse selection as in some other database query tools [CA96, CHMW96], the reason being that we believe that a somewhat trained person, such as a mediator/database administrator, is more efficient with typing OO queries than elaborate mouse selections. Future studies should investigate how to combine our text based interface with a graphical query language, e.g. along the lines of Pesto [CHMW96].

In the query editor AmosQL queries can be entered textually. A history of queries makes it possible to browse previously stated queries to edit or redo them. We do not save the query results in the history for memory and performance reasons. It is however possible to open new independent query editor windows for any query where a piece of the latest result is then visualized. Figure 4 shows an example of such a separate query window. This gives the user the choice to have several active queries and their results on the desk top to be used for inspection, copying, and pasting between query results and browsed objects.

The query editor is divided into three parts. First there is a label that states a numeric query history identifier. Then there are buttons to move backward and forward in the history and for help and (re-)execution of the current query.

Below the button panel there is a text area where queries can be entered textually or edited. All Windows standard formatting key shortcuts for text fields apply here and with a right click a standard menu pops up with cut, copy, paste, delete and select all. Typing F1 displays a help dialog with all available shortcuts. This is interfaced with the standard Windows clipboard, which means that it is possible to copy text to and from other programs in the same Windows environment. As usual in Windows, pressing CTRL-C in any window copies a string representation of the selected object to the standard system clipboard. This string can be pasted back into GOOVI as part of a query or even into a text input dialog in a completely different program.

To execute the query one can either press the *execute* button or type CTRL-ENTER. Further below there is an outliner that displays the result when the 'execute' button is pressed.

The result of an AMOS II database query consists of a *scan* of objects where the objects are visualized by red icons, as in Figure 2 and 3. For objects having names the browser maintains the correspondence between the OID and its name and the object's name is displayed as in Query#1 of Figure 2. An object is given a name by having an AmosQL function *name* defined. The user can define own *name* functions for any database type (class) to allow for customized display of OIDs. The results of queries are also connected to the clipboard for general cut-and-

paste between different GOOVI windows.

The result of a query can also be a scan of tuples, which are then visualized graphically to indicate their structure, as shown in Figure 4. Each tuple can have sub-tuple nestings to any level. Such a scan is visualized as a tree. If there are sub-tuples inside the elements of a scan special *collection icons* are created. This gives a nice presentation of most data structures retrieved from AMOS II database.

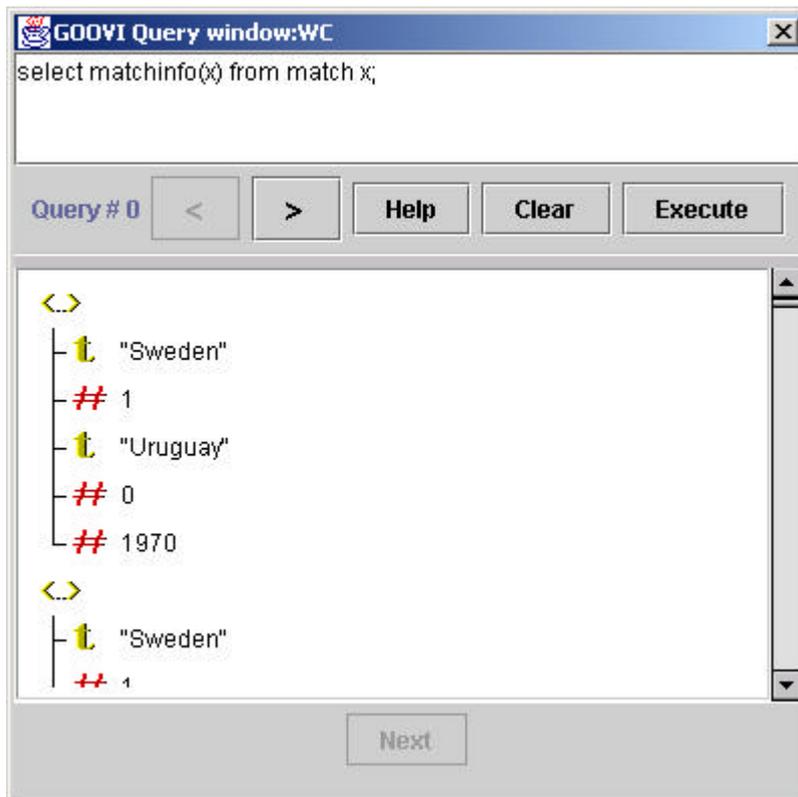


Figure 4 Scan visualization.

Fig. 4 shows the result trees from a query that returns rows of tuples containing string and integer elements. The collection icons (indicated < . . >) indicate tuples. If one of the elements in the tuple in the example had been a vector (ordered collection), e.g. if the two countries were put in a vector this would result in clickable nodes in this position.

This outliner is streamed and only a predefined (default 20) set of result elements are displayed initially. To get the next set of elements in the scan the 'Next row' button is used located at the very bottom of the query editor.

3.3 The Object Inspector

The object inspector window is opened if the user double clicks on an icon for an Amos II object. The object inspector allows displaying and editing the values of the attributes of the selected object.

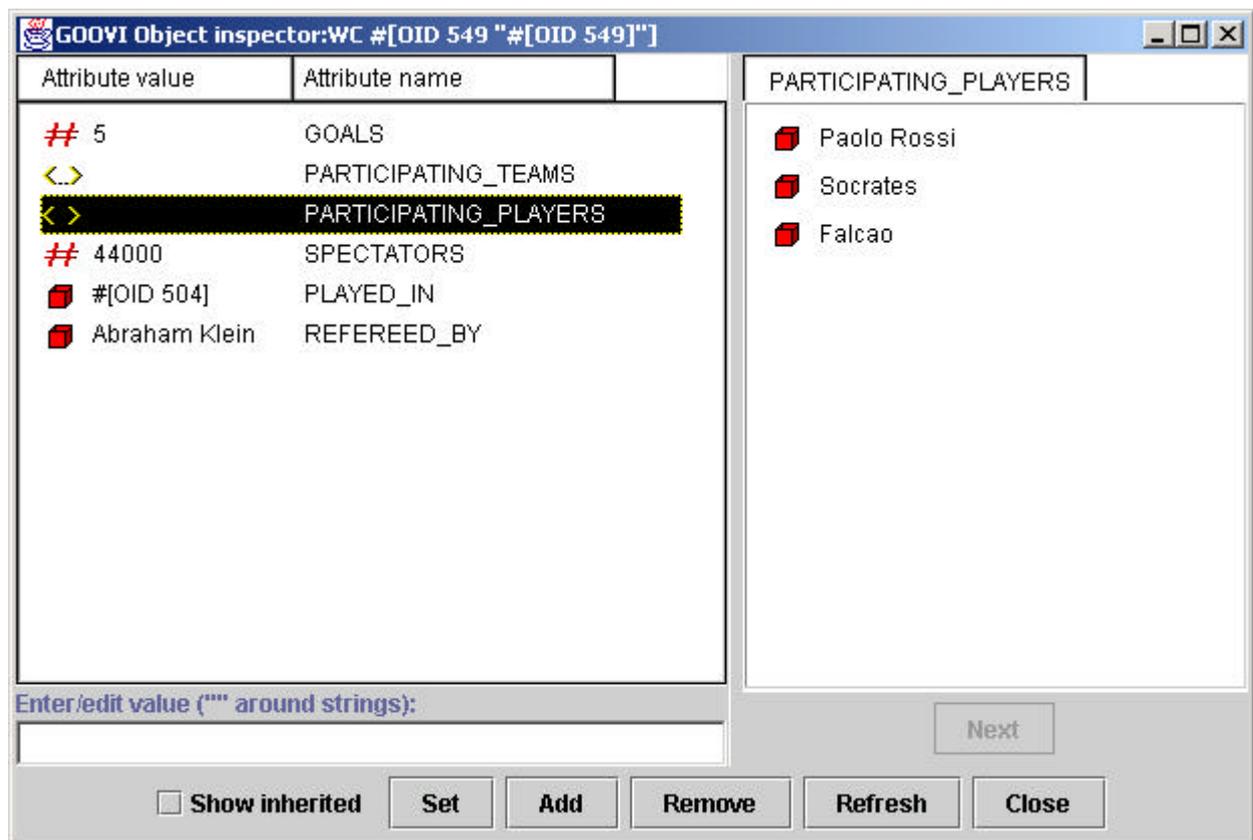


Figure 5 Object Inspector

Fig. 5 is an example of the object inspector in GOOVI. The *attribute* column displays the result of the function named in the *attribute-name* column.

The object inspector also allows updates of database attributes. By double clicking on attributes it is possible to enter new values. Cut-and-paste of objects between windows is, of course, supported.

Collection attributes are displayed as icons. As collections can be large, they are not immediately displayed, but retrieved by double-clicking on a collection icon. The collection is then displayed as a stream in groups of a predefined number of elements (usually 20).

3.4 The Function Inspector

The *function inspector* allows to view/edit arguments, results and source code of the definitions of Amos II functions (Fig. 6). These definitions are stored in the database as (meta-)objects of type *FUNCTION* and are retrieved by the function inspector through queries returning such meta-objects and their properties. Amos II functions can be of several kinds and the inspected function's type is indicated at the bottom of the function inspector window.

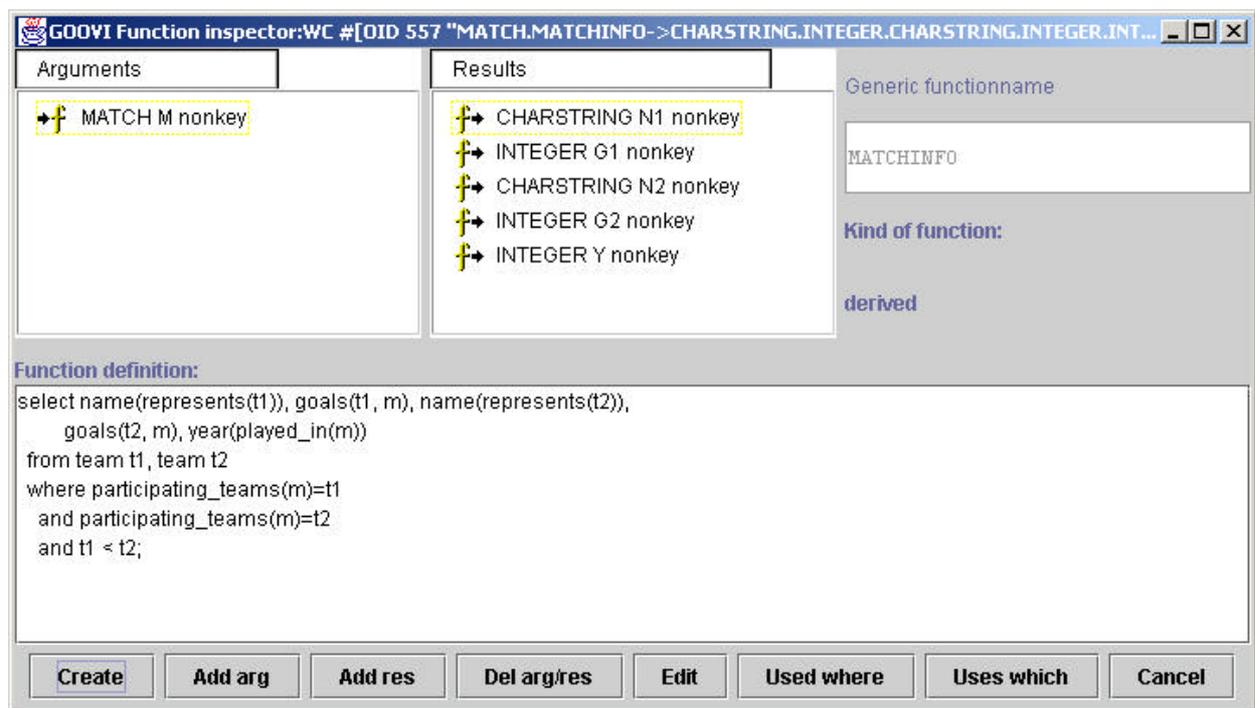


Figure 6 Function Inspector

The example illustrates a *derived function* defined in terms of other AmosQL functions through an AmosQL query. Functions can also be stored in the database or defined in some external programming language [RJ00]. Functions can also be overloaded on different types in which case a dialog lets the user choose which one of the resolvents to inspect. Roughly speaking attributes in OO languages correspond to stored functions, while methods correspond to derived and foreign functions.

4. Database integration with GOOVI

In order to combine data from different mediators, GOOVI allows importation of meta-objects, such as types (classes) and functions (method, attributes), from one AMOS II server to another. This is achieved by first selecting type and function definitions for exportation from one mediator and then importing them to another one.

To demonstrate the database integration support of GOOVI, assume we are in a tourist office and want to book flights. We have access to two databases, a British one called *HOLIDAYFLIGHTS_INC* and a Swedish one called *FLYGRESOR_AB*. The first database has a type called *FLIGHT* with the attributes: *flight_no*, *price*, *origin* and *destination*. The price is given in British pounds. The latter database has a type called *FLYG* with the attributes: *flyg_no*, *pris*, *start* and *destination*. *Pris* holds the price in Swedish crowns.

The first step is to import these types into the mediator *TORE*. We first have to select them for exportation from the mediators *HOLIDAYFLIGHTS_INC* and *FLYGRESOR_AB*. We therefore open a type browser for the *HOLIDAYFLIGHTS_INC* database and mark the type *FLIGHT*. Then we choose FILE->EXPORT in the menu bar. A dialog will confirm that the type is now ready for exportation. Next we go to the type browser for the mediator *TORE* and choose FILE->IMPORT. A dialog will confirm that the type was imported and it is called *FLIGHT@HOLIDAYFLIGHTS_INC* and automatically placed under type *USEROBJECT*. The same procedure is then repeated to import the type *FLYG* from the mediator *FLYGRESOR_AB* and then we have the situation shown in Fig. 7.

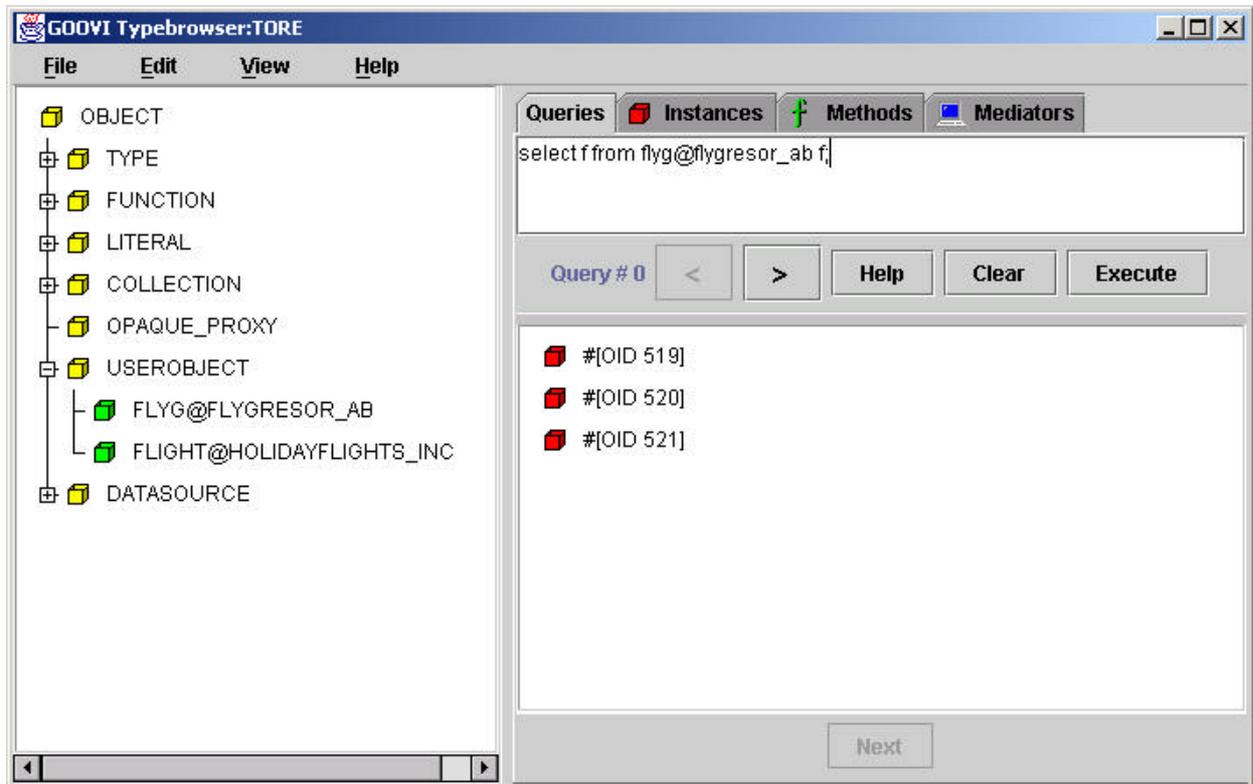


Figure 7 Local database after imports

An imported type can be used in database queries as other types, as also illustrated in Figure 7. The instances of an imported type are represented as *proxies* describing its origin. If more than one type is imported it is even possible to state multi-database queries and views that join data from different mediator databases. However, OIDs are unique within each mediator and such joins can therefore not compare OIDs but have to join on literal properties only.

To combine overlapping and conflicting data from several mediators, an integration union type, IUT, [JR99a] can be defined. IUTs define types whose extents are proxies of objects from other mediators and where the same entities may be found in more than one source but with different, and sometimes conflicting, properties.

To illustrate how IUTs are defined with GOOVI, we mark the two imported types in the mediator *TORE* and choose FILE->NEW->INTEGRATION TYPE. In Fig. 8 we see the so created *create integration* type dialog.

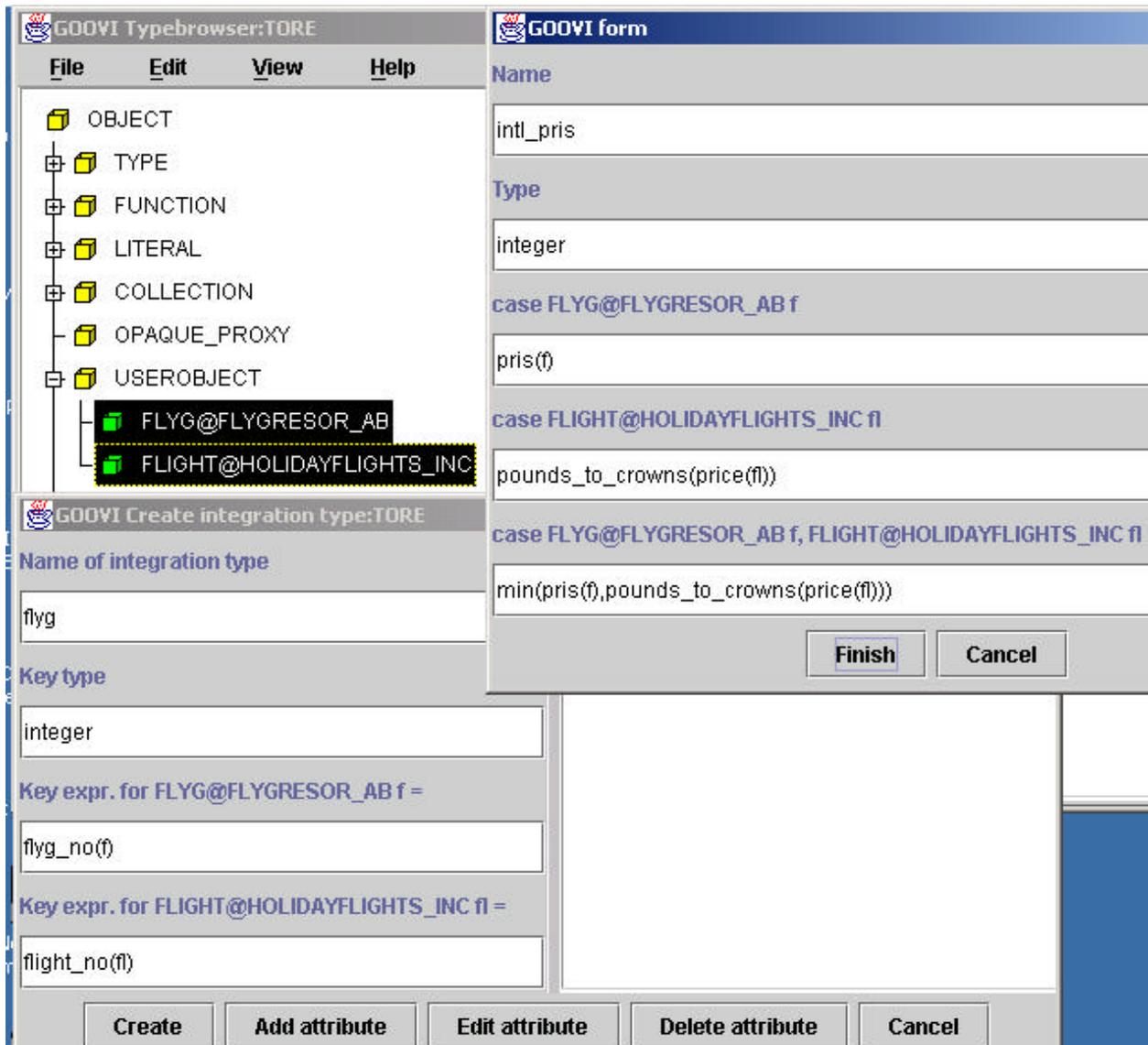


Figure 8 Integration Union Type creation

The first two fields in the IUT creation dialogue to left are the name and the key type of the IUT identifying equivalent objects from different mediators. In our example two objects are considered equivalent if their flight numbers (represented as integers) are the same. The flight number for *FLYG@FLYGRESOR_AB* is defined by the function *flyg_no* while the corresponding function for *FLIGHT@HOLIDAYFLIGHTS_INC* is *flight_no*. The next fields state the *key expression* for the two types to integrate which map instances of the integrated types to the common key. GOOVI generates unique variable names to be used in the integrating expressions. We specify there that the *flyg_no* of the mediator *FLYGRESOR_AB* corresponds to *flight_no* in *HOLIDAYFLIGHTS_INC*.

To specify how to define object attributes of the IUT in terms of attributes from different

mediators, one needs to define the attributes of the IUT by pressing ‘add attribute’ and filling in the popup form seen to the right in Fig. 8. Here the attribute *intl_pris* of the IUT is derived from the prices in *FLYGRESOR_AB* and *HOLIDAY_FLIGHTS_INC*. Different expressions are used if a flight exists in one of the two databases or in both. In this example the lowest price is chosen if the flight exists in both databases. The function *pounds_to_crowns* is a function defined in the mediator to convert British pounds to Swedish crowns. The attributes *name*, *origin*, and *destination* are defined in the same way. Finally ‘create’ is pressed to create the IUT *PRIS* in mediator *TORE*. It can then be browsed and queried as any other type. Updates of IUTs are currently not allowed.

5. Implementation

GOOVI is implemented in Java using a modern interface building tool, JBuilder 3 [Bor99]. Sun’s Java Native Interface JNI [SUN99] is used to tightly connect the Java Virtual machine (VM) to the AMOS II mediator database engine, which is written in C. The interface is completely based on sending Amos II queries and function calls to the database engine for execution. This is possible since all system objects (including types and methods) are represented as database objects that can be queried using AmosQL. While end users usually query the contents of the mediator databases, GOOVI mainly submits to the Amos II kernel meta-queries about the structures of mediators in a federation. AmosQL update statements are sent to Amos II when the user creates or updates objects, while AmosQL meta-data definitions are sent when new types and functions are created.

Program errors in Java applications, such as GOOVI, cannot crash the Amos II kernel system. The reason is that Java applications run in a separate thread from the kernel and the Java VM traps all Java program errors. Furthermore, the high level AmosQL based interface from Java prohibits calls to the Amos II kernel that can adversely affect it.

The Amos II kernel in its turn has TCP/IP based primitives for efficient communication with the name server and other mediator servers in the federation. Several novel techniques [JR99a, JR99b, JR00, RJ00] are used for efficiently executing multi-mediator queries over several distributed mediator servers. The techniques are based on installing optimized query plans on different mediators and then shipping bulks of data between them. For good performance we have therefore chosen to let GOOVI (and other Java applications) to communicate with the mediators in a federation through the local database rather than to directly communicate with each mediator.

Figure 9 illustrates how GOOVI is interfaced with a local Amos II database system and how the

system communicates with Amos II servers in the federation through the local mediator database system.

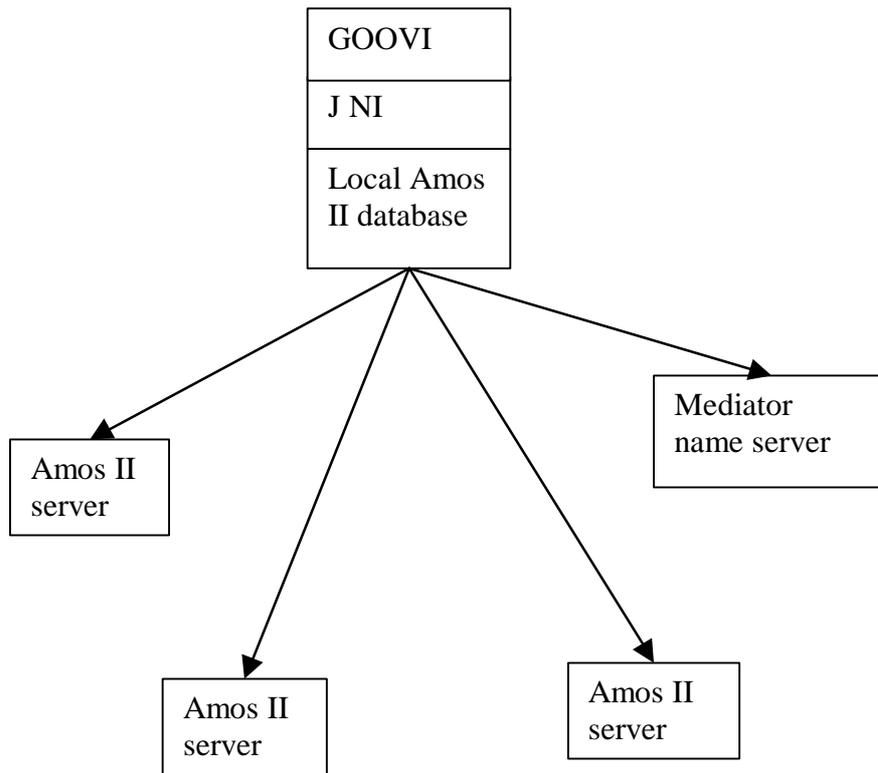


Figure 9: Connecting GOOVI to Amos II mediator databases

A possible disadvantage with the current approach is that some Amos II kernel code must be installed on the machine where GOOVI runs in order to provide the C-based interface to the Amos II kernel. However, we are working on a pure Java client-server interface to the local Amos II database, which would make it possible to run GOOVI as a Java applet, at the expense of much higher communication costs between GOOVI and the local database.

6. Conclusions

The architecture and appearance of the OO multi-mediator browser GOOVI was described. The purpose of GOOVI is to be able to browse, query, and integrate federations of

autonomous mediator servers in a computer network. The data integration facilities allow the definition of OO views that integrate data from several mediator servers. Each mediator server is autonomous and there is no global conceptual schema of all mediators in a federation. Instead a meta-mediator, the *mediator name server*, knows locations, names and some other meta-data about the members in a federation. The interface between GOOVI and a federation of mediators is implemented by internally querying the mediator name server for meta-properties using an OO query language and then use these meta-properties to connect to individual members in the federation. All communication between the browser and the mediator engine is thus made through the query language. This is possible since meta-objects are first class objects too.

The user can open a group of windows connected to a type browser window for each selected member of a mediator federation. In these windows type structures are investigated, queries are specified, and integrating OO views are defined. Database objects returned from queries are displayed graphically and can be browsed and used in other queries, etc.

GOOVI is the first OO multi-database browser that addresses graphical integration of multiple mediators. Such a tool is very useful for the mediator developer when integrating data from a federation of databases wrapped in mediators.

In summary, GOOVI has the following unique properties:

- It is an OO multi-database browser where several OO databases can be browsed separately.
- It can graphically integrate data from multiple distributed mediators in a federation.
- It respects the autonomy of the mediators in the browsed federation; it is thus not based on the availability of any global conceptual schema.
- Every object on any level in the federation is transparently inspectable, including mediators, type definitions, function definitions, and the database contents.

A limitation with the current GOOVI version is that many mediator definitions, e.g. queries and view properties, are not entered graphically, but as editable text string. Some system, e.g. Pesto and Jasmine, makes e.g. query specification fully graphical. Sometimes text is more convenient to enter than graphical interactions. This issue should be investigated further even though it does not alter the main principles of the system, which is the focus of this paper.

The use of Java as an implementation language provides a portable implementation and a rich library of user interface primitives. A uniform query based Java application program interface provides a flexible and high-level interaction between GOOVI and the Amos II kernel. The system is fully implemented under Windows 98/NT/2000 and downloadable from <http://www.dis.uu.se/~udbl/amos>. A Unix version of the Amos II kernel is also available with which Windows-based GOOVI clients can communicate with Unix based mediator servers.

7. REFERENCES

- [AGS90] R.Agrawal, N.Gehani, J.Srinivasan: ODE-View: The Graphical Interface to ODE. *Proc. ACM SIGMOD Conf.*, May 1990.
- [B99] L.Bouganim, T. Chan-Sine-Ying, T-T.Dang-Ngoc, J-L.Darroux, G.Gardarin, F.Sha: MIROWeb: Integrating Multiple Data Sources Trough Semistructured Data Types. *Proc. 25th Intl. Conf. On Very Large Databases (VLDB'99)*, Edinburgh, Scotland, 1999.
- [BE96] O. Bukhres, A. Elmagarmid (eds.): *Object-Oriented Multidatabase Systems*. Prentice Hall, 1996.
- [Bor99] Borland corporation: *Jbuilder 3 Developer's Guide*, 1999
- [CA96] Computer Associates corporation: *Jasmine Version 1.2 Tutorial*, 1996.
- [CHMW96] M.Carey, L.Haas, V.Maganty, J.Williams: PESTO: An Integrated Query/Browser for Object Databases. *Proc. 22nd Conf. On Very Large Databases (VLDB'96)*, 203-214, 1996.
- [F89] D.H.Fishman, J.Annevelink, E.Chow, T.Connors, J.W.Davis, W.Hasan, C.G.Hoch, W.Kent, S.Leichner, P.Lyngbaek, B.Mahbod, M.A.Neimat, T.Risch, M.C.Shan, W.K.Wilkinson: Overview of the Iris DBMS, in W.Kim, F.H.Lochofsky (eds.): *Object-Oriented Concepts, Databases, and Applications*, ACM Press, 1989.
- [HKWY97] L. Haas, D. Kossmann, E.L. Wimmers, J. Yang: Optimizing Queries across Diverse Data Sources. *23rd Intl. Conf. on Very Large Databases (VLDB'97)*, 276-285, 1997
- [G97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y.Sagiv, J. Ullman, V. Vassalos, J. Widom: The TSIMMIS Approach to Mediation: Data Models and Languages. *Intelligent Information Systems (JIIS)*, Kluwer, 8(2), 117-132, 1997
- [JR99a] V.Josifovski, T.Risch: Functional Query Optimization over Object-Oriented Views for Data Integration. *Journal of Intelligent Information Systems (JIIS)*, Vol. 12, No. 2-3, 1999
- [JR99b] V.Josifovski, T.Risch: Integrating Heterogeneous Overlapping Databases through Object-Oriented Transformations. *25th Conf. on Very Large Databases (VLDB'99)*, 435-446, 1999.
- [JR00] V.Josifovski, T.Risch: Query Decomposition for a Distributed Object-Oriented Mediator System . To be published in *Distributed and Parallel Databases J.*, Kluwer, 2000.
- [MDT88] A.Motro, A.D'Atri, L.Tarantino: The Design of KIVIEW: An Object-Oriented

Browser. *Proc. 2nd Intl. Conf. On Expert Database Systems*, April 1988.

[LP97] L.Liu, C.Pu: An Adaptive Object-Oriented Approach to Integration and Access of Heterogeneous Information Sources. *Distributed and Parallel Databases*, Kluwer, 5(2), 167-205, 1997.

[RJ00] T.Risch, V.Josifovski: Distributed Data Integration by Object-Oriented Mediator Servers. To be published in *Concurrency - Practice and Experience J.*, John Wiley & Sons, 2000. (Available at <http://www.dis.uu.se/~udbl/publ/concur00.pdf>)

[RJK00] T.Risch, V.Josifovski, T.Katchanouov: *AMOS II Concepts*, Department. of Information Science, Uppsala University, 2000, (available at <http://www.dis.uu.se/~udbl/amos/>).

[SUN99] Sun corporation: JNI - *Java Native Interface*, 1999
(<http://www.javasoft.com/products/jdk/1.1/docs/guide/jni/index.html>).

[TRV98] A. Tomasic, L. Raschid, P. Valduriez: Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Date Engineering*, 10(5), 808-823, 1998

[Wie92] G Wiederhold: Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3), 38-49, 1992.