CORBA-BASED DATA INTEGRATION FRAMEWORK

Antonis Ramfos[1], Ralph Busse[2], Nikos Platis[1], Peter Fankhauser[2]

[1]INTRASOFT S.A.	[2]GMD-IPSI
2, Adrianiou Str.	Integrated Publication and Information
11525 Athens, Greece	Systems Institute, Dolivostraße 15,
ramfos@isoft.intranet.gr	64293 Darmstadt, Germany
platis@intrasoft.gr	{busse,fankhaus}@darmstadt.gmd.de

ABSTRACT

The evolving developments of the recent decades in information management have resulted in a situation where information is stored and managed by a large variety of systems around the globe. The growing requirement for information support to the globalised social and economic forces makes efficient and effective availability of distributed information and the correlation of relevant data a pressing business need. The most recent CORBA standard of the Object Management Group provides standardised interfaces for accessing remote data but the task of collecting and correlating the relevant, heterogeneous sources is still left over to the human user. The presented work proposes a CORBA-based *Data Integration Framework* with which data is integrated and offered to users according to their application needs. This framework is composed of two CORBA horizontal facilities, the design-time *Object Composition Facility*, which supports the data integration process, and the run-time *Composed Access Facility* which, based on the previously defined data integration, performs all required distributed accesses transparently to the users.

1. INTRODUCTION

Since 1950s businesses developed a voracious and continuously increasing appetite for information. We are now faced with the situation where organisations have collected oceans of data that is managed in file systems, commercial application packages with their own 'DBMSs', or in one of the numerous DBMS products of a variety of types.

As the world is becoming highly interconnected and is moving towards a single market, standardised access of heterogeneous pre-existing data sources takes high priority with cross-industry businesses. According to the distributed computing paradigm, the Object Management Group (OMG) defined the *Common Object Request Broker Architecture (CORBA)* as a standard for implementing distributed client/server applications (OMG, 1991). An Object Request Broker (ORB) is introduced in order to tie distributed objects together. The ORB provides the means to locate and activate objects on a network, regardless of operating platform or implementation language. By standardising the network interfaces at the client and server sites the CORBA framework maximises portability, reusability, and interoperability, and has become the de-facto industry standard for communication middleware.

Although CORBA-based interconnected servers are very popular in today's distributed computing environments, the task of extracting and correlating information from disjoint presentations of relevant data is left over to the client applications. As organisations become more extended and distributed, it is not efficient for applications to deal with the complexities introduced by the correlation of multiple data and service sources. Rather, a dedicated integration middleware should be employed that efficiently integrates heterogeneous pre-existing data and presents it as a single, integrated system. Client applications are no longer aware of the distribution and all remote accesses are transparently performed by the middleware.

Many attempts have been made in the past in order to build integrated systems, that include the efforts in the late 1970s to build distributed database management systems (DDBMS) (Ceri et al., 1984), those that targeted to multi-database management systems (MDBMS) and federated databases in the 1980s (Landers et al., 1982; Litwin, 1985; Shet et al., 1990), and the efforts in the 1990s on the development of data warehousing for decision support applications (Immon et al., 1993; Widom, 1995; Gupta et al., 1996). Although the integration of heterogeneous pre-existing data sources has been a pressing business need, none of the above efforts has succeeded in becoming widely accepted due to the technical challenges of the problem. The technical challenges include, among others, performance considerations, concurrency control, preservation of local autonomy, support for unpredictable queries on volatile data, collaborative work, and integration of flat files, or other non-relational and non-data management systems. However, object-oriented multidatabase systems arose in response to the above challenges and have proposed solutions to a variety of issues in data integration (Bukhres et al., 1995).

The approach to data integration reported in the current work marries the latest object-based technology for

database integration from the ESPRIT IRO-DB project (Gardarin et al., 1997) with the CORBA standard in order to achieve integrated access to pre-existing data sources that exhibit not only data management heterogeneity, but also operating platform heterogeneity. This provides an achievable and realistic solution to the overall problem of data integration, where distinct CORBA objects that represent data in heterogeneous data sources can be integrated and accessed in a way that reflects the conceptual requirements of users and applications of today's integrated world.

Section 2 of this paper describes the overall architecture of the Data Integration Framework. Section 3 gives details about the design-time and run-time components and about the different transformation steps during data integration. In Section 4, we present our prototype, that has been implemented to show the feasibility of the approach and serves as a basis for further development, which is detailed in Section 5. A conclusion summarises the paper.

2. FRAMEWORK ARCHITECTURE

In addition to the core CORBA specification, the OMG defines standard interfaces and related functions for two supplementing layers. The lower-level *CORBAservices* provide basic support functionality for individual objects, including *Transaction Service*, *Event Service*, *Time Service*, *Query Service, Naming Service*, and many others (OMG, 1997). The intermediate-level *CORBA-facilities*, on the other hand, provide functionality for applications (OMG, 1995). While *horizontal* CORBAfacilities, such as compound document management, can be used by virtually every business, the *vertical* CORBAfacilities standardise management of information specialised to particular industry groups, e.g., Health Care.

The proposed Data Integration Framework consists of two horizontal CORBAfacilities with which distributed data in the form of CORBA objects can be integrated and subsequently accessed by cross-industry applications (see Figure 1): an *Object Composition Facility* and a *Composed Access Facility*. Another element of the Data Integration Framework is the *Local Data Adapter (LDA)* which scales up a standard CORBA Object Adapter's capability so that it is able to handle data sources that deliver large numbers of fine grained objects, e.g., records of a database system. Such LDAs are currently offered in the form of object database adapters by vendors who implement the CORBA standard, e.g., IONA's Orbix Database Adapter Framework.

The *Object Composition Facility (OCF)* is a stand-alone module that provides design-time support for the semantic integration of heterogeneous information servers. It consists of a graphical design tool, the *Composition Editor*, which reads IDL specifications of existing information servers and presents them to the designer as graphs. Based on these graphs, the designer can define correspondences and conversions between the different IDL interfaces. An underlying methodology uses this information for incrementally merging the local interfaces into an integrated view. It detects further correspondences and rejects inconsistencies. When an integrated view is reached, its corresponding IDL specification is constructed and the supporting C++ code for the specific Composed Access Facility is generated by the *Code Generator*. The approach taken for this facility capitalises on the expertise gained from the *Integrator's Workbench* developed for the design of integrated ODMG-views in the ESPRIT project IRO-DB.
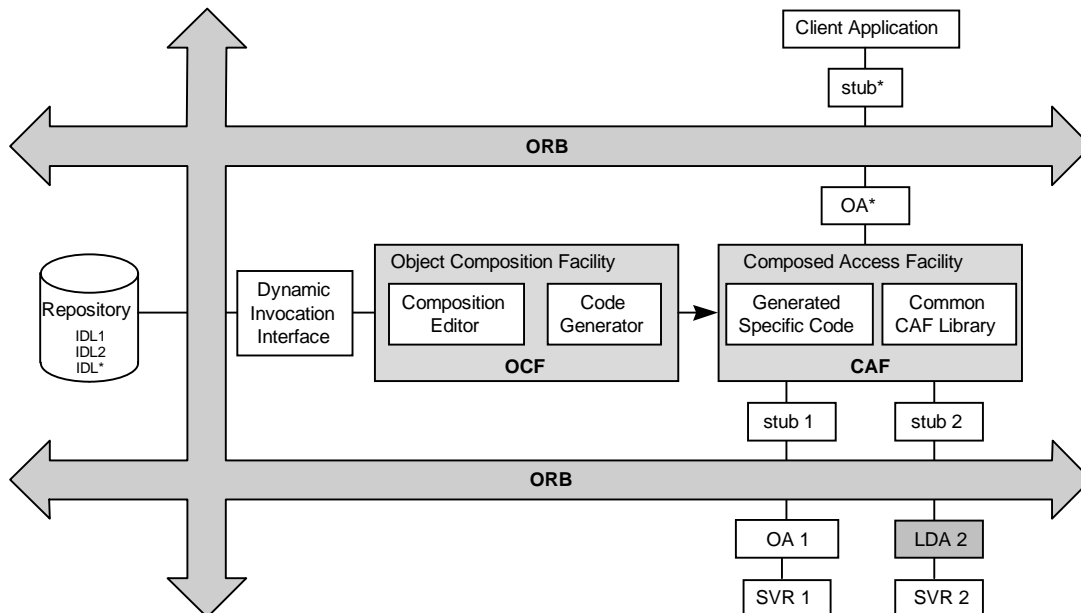
The *Composed Access Facility (CAF)* performs the

Figure 1: CORBA-based Data Integration Framework Architecture

instance composition at run-time. It consists of a common object management library that is extended with generated schema-dependant C++ code for the specific integration. Specifically, CAF manages surrogate objects representing compositions of local objects. The client application accesses the surrogate objects through corresponding integrated stubs. Accesses to the surrogate objects are transparently delegated to the original local objects on the servers. Parameter data and results are converted accordingly. The realisation of CAF is based on the current specifications of the relevant CORBA services, such as Query Service, Transaction Service, Concurrency Service, etc. In this way, a more robust and highly standardised approach to object management is achieved, than the one followed in IRO-DB and other data integration projects.

The operation flow in Figure 1 is as follows: Heterogeneous information servers (SVR1/2) export their functionality in the form of IDL specifications (IDL1/2). OCF uses the Dynamic Invocation Interface (DII) to read these IDL specifications from the ORB Repository and based on the designer's integration specifications, it generates a single, integrated IDL specification (IDL*) and additional C++ code that implements the composed objects in the CAF. The ORB itself generates further code fragments (stubs and skeletons) from the various IDL specifications. Together with generic Object Adapters (OA) these fragments establish the connection between client and server. The CAF acts now as a client of the information servers and accesses their information through the generated individual stubs. At the same time, it acts as a server to the actual client application providing it with a single, integrated IDL interface. Applications can hence access all the information through a single server without bothering about the different underlying information servers. In addition some Object Adapters can be replaced with specialised LDAs (LDA2).

The presented functionality can be directly compared with the MIND system developed at the Middle East Technical University in Ankara (Dogac et al., 1995 and 1996). There are, however, some fundamental differences that distinguish these systems from each other. The first difference is the set of supported information servers. The MIND system is explicitly designed to support the integration of relational and object-oriented databases, just like in the IRO-DB system. The presented Data Integration Framework, however, investigates the possibilities of integrating arbitrary information sources. Although database integration is definitely a starting point for our system, file-based servers and function libraries will be supported as well. Any information server that exports an IDL interface fits into the architecture. The second major difference is the object granularity. The restriction to database integration allows to employ coarse-grained objects, i.e. to represent each participating database as a single object and to perform all database accesses by sending query statements to these objects. Several advantages of this approach make it valuable in the database area (Dogac et al., 1996), but on the other hand it shows some restrictive

shortcomings. The system requires the existence of a query engine at each participating database which makes it impossible to integrate arbitrary information servers. Furthermore, the functionality of the integrated objects is restricted, because interaction with the distributed (fine-grained) objects is restricted to the capabilities of the query language. Finally, the objects cannot be smoothly integrated into the application programs. Due to these reasons, we are following the fine-grained object approach, where the integration is performed directly on the objects exported from each information server, according to its IDL specification.

3. FRAMEWORK DESIGN ISSUES
This section presents the design of the CORBA-based Data Integration Framework in more detail.

3.1 COMPOSED ACCESS FACILITY
Figure 2 shows the two parts of the CAF. One part consists of the *Generated Specific Code* which is specifically generated for each integration at design-time by the *Code Generator* of OCF. The other part is the library that contains the common parts of all CAF implementations (*Common CAF Library*).

3.1.1 GENERATED SPECIFIC CODE
The *Generated Specific Code* consists of a set of C++ classes, called *target classes*, that implement the integrated view that is exported to the clients. The classes are generated by OCF for a specific set of given information
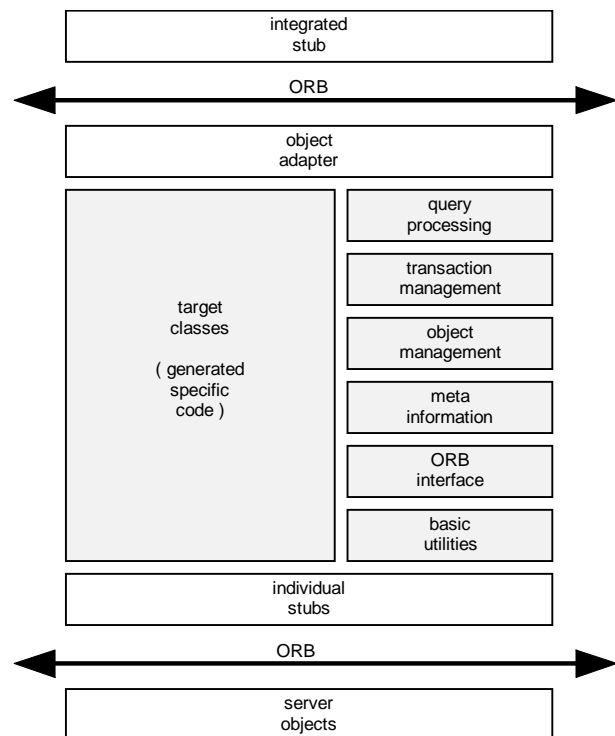

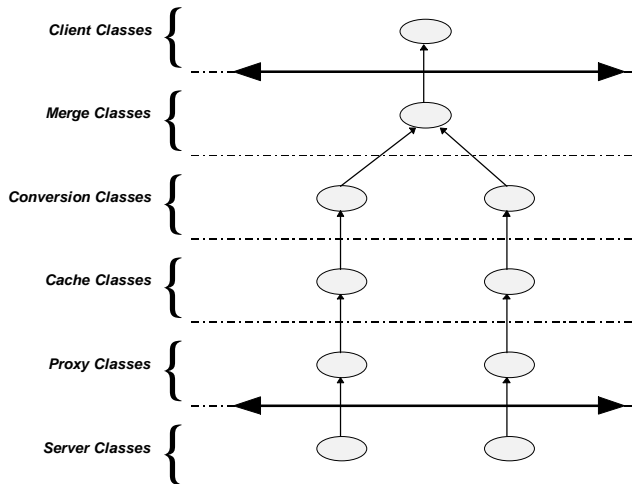**Figure 2: Composed Access Facility Architecture**

Figure 3: Target Class Hierarchy

servers and are organised in several levels, as shown in Figure 3.

Server Classes: The server classes do not belong to the CAF. They are the original classes that reside on the local information servers. They define the export interfaces and provide the server-side ORB access and the information server implementation.

Proxy Classes: CORBA replicates the interfaces of the server classes at the client site by generating for each server class a corresponding *proxy class*. The instances of proxy classes are a kind of 'intelligent pointers'. They establish the connection from the client program to the server objects by transparently delegating all accesses to the corresponding server object. The proxy classes are generated by the ORB's IDL compiler. Therefore, they do not belong to the CAF, but they form the basis for all other target classes.

Cache Classes: During design time, the administrator may create *cache classes* for selected proxy classes. The delegation behaviour of the proxy classes causes network traffic for each single attribute access. While this solution is appropriate for servers providing functional services, it constitutes a severe bottleneck for information servers, like, e.g., databases. To avoid unnecessary network accesses, a cache class can be linked between the application and the proxy class. Whenever a read access is performed, the result is kept in a local cache and can be reused for subsequent accesses. CORBA's strictly functional approach of using 'read' and 'write' methods for accessing an object's attributes allows for both object and attribute caching strategies. Explicit cache classes are intended as a generic replacement for proprietary cache mechanisms, like, for example, IONA's smart proxies.

Conversion Classes: The classes of the conversion level are the first classes with application-specific behaviour. They are responsible for restructuring the information that is provided by the proxy classes, the cache classes, or other conversion classes in order to

achieve homogeneous representation of the data. The IDL specification and the C++ implementation code for conversion classes is generated by the OCF.

Merge Classes: When the target classes of the information servers are sufficiently homogenised, related classes from different servers can be merged into a single class. In addition to further conversion operations, the *merge classes* must also identify corresponding instances from the different information servers. The strategy for combining the information is determined during the design process. In addition to the strict separation between conversion and merge, it is also possible to perform all conversions directly during the merge. This is only a question of complexity.

Client Classes: The target classes in CAF result in a class hierarchy that can be exported to client applications. This export is performed by registering the interfaces as a new CORBA information server. As with the registration of the local information servers, the IDL compiler generates proxy classes for the integrated IDL interfaces. These proxy classes are linked to the client application and establish the access to the CAF.

3.1.2 COMMON CAF LIBRARY

The six boxes on the right hand side of Figure 2 represent the common modules that can be directly reused in all CAFs. A *basic utilities* module implements support functionality like string conversions, error management, output streams, etc. An *ORB interface* module is used for simplifying the CORBA access. It contains global variables to hold ORB instances, and it customises calls to ORB functions and to the CORBA Naming Service. When non CORBA-compliant functions that are specific to the ORB product used are utilised, they are encapsulated in this module to provide for a better adaptation to other ORB products. The *object management* module contains common base classes for all target classes. It implements generic access to the objects' properties and methods at run time. This is necessary for supporting query evaluation. Furthermore, extent managers take care of object identity during sequential query evaluations on the same data.

The remaining modules cover *transaction management* and *query processing* and rely on the full functionality of the CAF. To perform their work, they need information about the structure and dependencies of the target classes. This information is provided in a *meta information* module. This module retrieves the basic structural information from the CORBA interface repository and extends it with mapping information from the integration process. Both the transaction and query module will define appropriate interfaces to interact with the corresponding CORBA services.

3.1.3 SERVER ENTRY POINTS

The CORBA standard defines the distribution of objects in the network. When an application executes a method on an object reference, the ORB is responsible for

4

locating the object implementation and for performing all necessary network actions to let it execute the desired function. This ORB mechanism, however, requires that the client has already hold of an object reference. For this purpose, CORBA defines a specific service for the identification of objects, namely the *Naming Service*. This service represents an object dictionary with the functionality of 'yellow pages'. An information server can register its objects under a distinct name in the global dictionary. Each client that knows the name can then retrieve the desired object reference from the global dictionary.

Information servers have a choice of how to register their objects with an ORB. With *exhaustive registration* an information server registers every contained object with the Naming Service. The client can directly access each registered object via corresponding mnemonic string names. On the other hand, if we consider that information servers may contain a large number of objects, it makes sense to register with the Naming Service and make available to clients a single object that constitutes the entry point to all objects contained in the server. We refer to this object as a *factory object.* Starting from the factory object, the client can then reach the complete collection of server objects by navigation. For this purpose, the factory object has to provide some attribute or method providing the necessary object references. Since we target at the integration of information servers that may contain a large number of objects we adopt the latter strategy where each information server and CAF registers exactly one entry factory object (*factory registration*).

3.2 OBJECT COMPOSITION FACILITY

The modules described above provide the run-time environment of the Data Integration Framework. A core element of the system, however, is the design-time support. The major problem with data integration is the detection of correspondences and similarities between different data representations. As long as the support for correspondences is restricted to the merger of attributes with identical or synonymous names, the design can be performed straightforwardly. Typically, though, data items do not correspond in such 1-to-1-relationships. The attributes may represent overlapping real world semantics, or one attribute may generalise the other one. They may have a common generalisation, or they may integrate in a 1-to-n-correspondence. Even when the real world semantics is identical, their extents may be identical, overlapping, or disjoint. Furthermore, data items are typically highly interconnected and it is necessary to perform integration steps on the connecting paths. All these tasks cannot be performed automatically. They require interaction with a human designer. Even more, the designer will be overloaded with such a complex task and needs support from the system.

In IRO-DB, a graphical design system has been implemented, that supports a designer in the task of database integration, the *Integrator's Workbench*. Database schemas are presented as graphs on the screen and can be manipulated by the designer. The initial detection of correspondences and the lossless semi-automatic computation of an integrated schema is performed under the guidance of a powerful methodology. It detects inconsistencies and presents possible correction alternatives. When the design task is completed, a code generator creates both the specification of the integrated schema and the implementation code that performs the object integration at instance level.

The Object Composition Facility is an adaptation of the Integrator's Workbench to the CORBA environment. The underlying algorithms are taken over, but the data model is converted to the OMG data model and the single-object design of CORBA is reflected. In addition, the code generator must be tailored to account for the new environment, generating code that fits the CAF requirements.

4. AN EXAMPLE COMPOSED ACCESS FACILITY

Based on the presented design structure, we have implemented a first reduced CAF that achieves the integration of two example information servers. For this implementation we used a popular ORB product, that of IONA's Orbix. Below we summarise the implementation of both the example servers and the corresponding CAF.

The example application built involves two local information servers that contain people objects. The first server contains a class named *Employee* that keeps people's employment details. The second server contains a class named *Person* that keeps people's personal information. Data of the local servers is stored in text files, and basic data management operations (creation, retrieval, update, deletion) have been implemented on these files. Following the design principle, the CAF acts as a third server, that contains a class named *Staff*, which composes the local information into an integrated class. This integrated class merges all attributes of the local objects together, and provides transparent access to the local objects' attributes and methods by propagating calls to and from the local servers.

The schema of each local information server is exported to clients as an IDL interface specification. This IDL interface specification includes methods for the update and deletion operations. A separate IDL interface is defined in order to serve as the object factory of the server. The object factory comprises methods to retrieve and create objects, complementing the data management methods defined in the exported IDL interface. On the other hand, the CAF is similarly registered via two IDL interfaces. The first includes the *integrated schema* accessible by client applications, as well as methods for the update and deletion operations for the integrated objects. The second interface corresponds to the object factory of the integrated objects, and provides methods for the creation and retrieval operations of the integrated objects. The separation of the data management methods between the two IDL interfaces is due to the fact that an IDL definition cannot provide for class methods that can be executed without a receiver object, (i.e., static methods). Methods

supported by IDL interfaces can only act upon already referenced objects.

The IDL interfaces `Employee` and `Person` of the example application hold the Employee and Person local schemas, respectively, and provide the update and deletion methods. Here are the IDL definitions:

```
interface Employee  {
    attribute string           name;
    attribute string           job_title;
    attribute unsigned short   phone_extn;
    attribute unsigned long    salary_drs;

    void save() raises (OCS::Exception);
    void remove() raises (OCS::Exception);
};

interface Person  {
    attribute string           name;
    attribute string           address;
    attribute unsigned long    phone;

    void save() raises (OCS::Exception);
    void remove() raises (OCS::Exception);
};
```

Interfaces `EmployeeFact` and `PersonFact` provide access points for `Employee` and `Person` objects, respectively. Each has just two member functions, to create and retrieve `Employee` and `Person` objects. Their IDL definitions are as follows:

```
typedef sequence<Employee> EmployeeSeq;

interface EmployeeFact  {
    Employee newEmployee(in string name)
        raises (OCS::Exception);
    EmployeeSeq getEmployee(in string name)
        raises (OCS::Exception);
};

typedef sequence<Person> PersonSeq;

interface PersonFact  {
    Person newPerson(in string name)
        raises (OCS::Exception);
    PersonSeq getPerson(in string name)
        raises (OCS::Exception);
};
```

The IDL interface `Staff` holds the integrated schema that represents the integration of Employee and Person local objects unified by their name, and provides the update and deletion methods. The IDL interface `StaffFact` provides access to objects through methods to retrieve and update such objects. Both interfaces follow the pattern of the Employee and Person interfaces.

```
interface Staff  {
    attribute string           name;
    attribute string           job_title;
    attribute string           address;
    attribute unsigned long    home_phone;
    attribute unsigned long    work_phone;
    attribute unsigned long    salary;

    void save() raises (OCS::Exception);
    void remove() raises (OCS::Exception);
};
typedef sequence<Staff> StaffSeq;

interface StaffFact  {
    Staff newStaff(in string name)
        raises (OCS::Exception);
    StaffSeq getStaff(in string name)
        raises (OCS::Exception);
};
```

Each of the IDL interfaces presented above is implemented in C++ by a corresponding class. As mandated by the CORBA standard, these implementation classes con-

tain public methods to access the IDL interfaces' attributes and to realise their methods.

The classes that implement the local IDL interfaces have private data members that hold the schema's attributes; additional data members may hold any attributes that are required for implementation purposes, such as appropriate object identifiers of local objects. Related data management operations are implemented by direct calls to the corresponding operations of the local information servers. The implementation class of the object factory interface does not need any private data members, and it simply comprises functions implementing the operations of this interface.

On the other hand, the classes that implement the integrated IDL interface need only hold references to the proxy objects that compose the integrated object. In order to get and set the attributes of the integrated object, the corresponding attributes of the proxy object(s) are accessed on the fly, taking care of any necessary conversions. Similarly, data management operations at the integrated level are realised by propagating calls to the local servers. In the same fashion, the implementation of the integrated object factory class needs references to the proxy factories, to which it delegates the appropriate calls.

In the current implementation, all specific code was produced manually. In the completed version of the Data Integration Framework, the user will be able to specify integrated views and their dependencies on the local objects with the help of the graphical Object Composition Facility, which will automatically generate all the required specific code, as previously presented.

5. FUTURE EXTENSIONS - QUERY SUPPORT

The current implementation of the Data Integration Framework is based on the core CORBA functionality which is directed towards single-object accesses. Information servers export single objects and clients access these objects through references. This mechanism is rather restrictive and will be extended with support of object collections and with declarative query functionality. This provides more functionality to the end user and it allows for a more general specification of the composed objects.

There are two Object Services in the CORBA standard that directly influence the development of object collection support in our framework: The *Object Collection Service* and the *Query Service*.

5.1 OBJECT COLLECTION SERVICE

The *Object Collection Service* (IBM, 1996) is still under development and not yet contained in the official COS Specification (OMG, 1997). The service provides a comprehensive collection of aggregate data types. Based on a root interface named *Collection*, a complete hierarchy of ordered and unordered sets, bags, and maps as well as stacks and queues are defined. Each collection object is designed to manage a homogeneous aggregation of elements of any object or data type. It provides operations for

inserting, retrieving, replacing, and removing elements. For consecutive access to the elements, a hierarchy of *iterators* is defined, and for the creation of new collections, a corresponding hierarchy of *CollectionFactories* is contained in the specification. The functionality of this service can be compared with the *ODMG collections* (Cattell, 1996), the *C++ Standard Template Library STL* (Musser et al., 1996), or the *Rogue Wave Tools.h++* toolkit (SunSoft, 1995). Until the Object Collection Service is available for the Orbix ORB, we are using the Rogue Wave toolkit for further development.

5.2 QUERY SERVICE

A more relevant service is the *Query Service*, which is already contained in the COS Specification Document (OMG, 1997). It defines the processing of database-like query statements. Only the evaluation mechanism, i.e. the framework, is specified in the standard. The query language and all internals are left over for the service implementor. The query service specification contains also a collection model to represent aggregated data. As soon as the Object Collection Service is accepted, its collection classes will be used instead.

Although the query service is not restricted to a specific query language, a compliant service implementation is required to provide either SQL-92 (ANSI, 1993) or OQL-93 (Cattell, 1996) evaluation. In addition, any proprietary language can be supported by the system. The query evaluation is based on a *QueryEvaluator* object. Given a query string in a supported language, the QueryEvaluator is responsible for performing the complete query evaluation returning a result to the caller. The class has explicitly been designed to delegate parts of the evaluation to other QueryEvaluator objects. A special type of evaluator is the *QueryableCollection*. Such a collection evaluates a query on its elements, taking into account the knowledge about the elements' type. When the elements are themselves derived from the QueryEvaluator interface, the evaluation is automatically delegated. For more complex queries that need some global management, a *QueryManager* object is used. The QueryManager interacts with objects of type *Query*, which independently evaluate subqueries. The provided implementation liberty together with the delegation policies allow for optimised evaluators for different environments. For example, an evaluator class for a database management system can exploit the internal indexing and iteration facilities of the database.

5.3 ACCESS EXTENSION LEVELS

Currently integration is performed by handling explicit references to individual objects. The registered factory object is retrieved from the Naming Service, further objects are accessed by navigating from this starting point, and composed objects are generated carrying these explicit references. The composite objects are then registered with the Naming Service to provide clients with the initial entry point. This layout of combining single objects to new objects is shown in Figure 4.
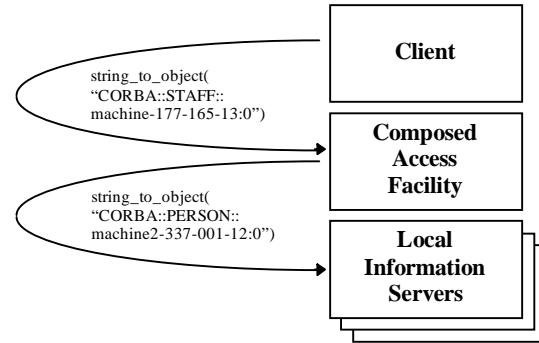

Figure 4. Single Object Access

The further development is dedicated to the inclusion of query capabilities into the system. Employment of Query Services allows to replace the navigational object access via Naming Service with declarative object lookup. Furthermore, it allows to extend the functionality of the system by supporting retrieval of object collections. This extension will be performed in two phases. The first phase is to exploit the query service capabilities of the underlying information servers. Instead of hard-coding single-object accesses into the composed access facility, a declarative query string can be attached to the composed objects. The original objects are then retrieved by evaluating the query on the information servers instead of retrieving them from the Naming Service. This situation is shown in Figure 5.
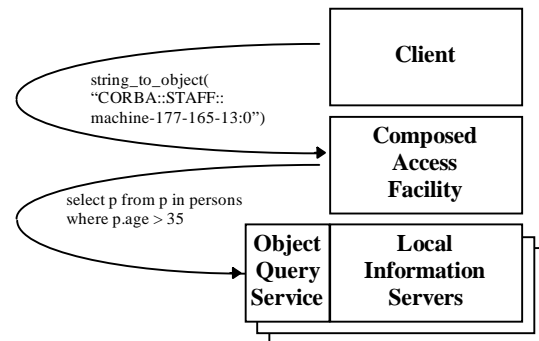

Figure 5. Object Access Using Query Service

In this first extension phase the CAF itself does still only provide a single-object interface to the clients. Therefore, the second level of extension will be to provide query capabilities to the client, i.e. on top of the Composed Access Facility. The client may submit queries to the CAF to retrieve composed objects, and the CAF will itself decompose and reformulate the queries to get the necessary data from the information servers. This complete query support is shown in Figure 6.

This approach is very well supported by the above-mentioned delegation feature of the QueryEvaluators. Finally, there will be a hierarchy of specialised Query-Evaluators as shown in Figure 7.
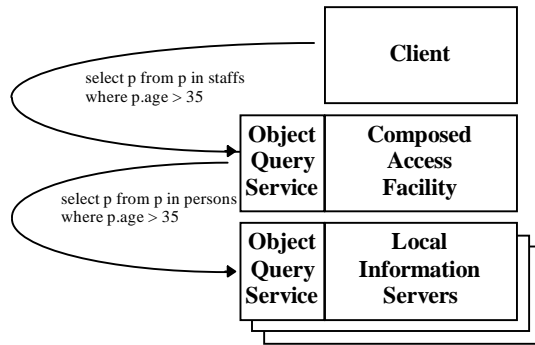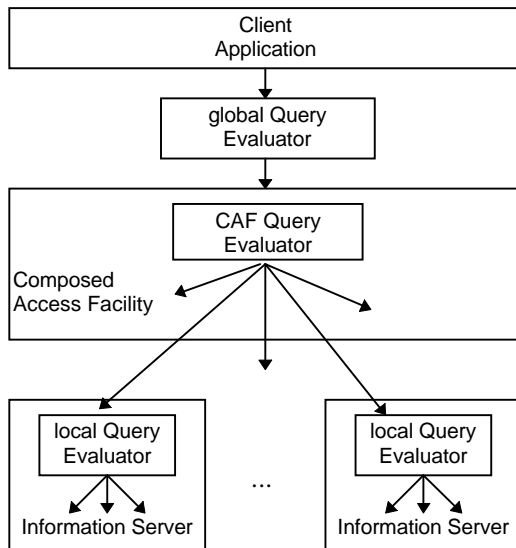
Figure 6. Full Query Access

Figure 7. Multi-Level Query Evaluation

6. CONCLUSION

We have shown, that it is possible to set up a framework in the CORBA environment that allows for transparent data integration for client applications. The direct, object-wise implementation required only very little support from the Common CAF Library. Most of the common modules need to be filled as soon as collection support requires concurrent, generic object accesses. The future development will be divided into three phases. The first step is to perform an availability analysis of query service support. It is very likely, that no implementations will be available in time. In this case, at least a minimal QueryEvaluator must be implemented by ourselves. As soon as such a service is available, the second phase will exploit collection-based accesses to the local Information Servers without providing such functionality to the client application. The complete query support on all levels will be implemented in the third phase of extensions.

7. REFERENCES

ANSI, 1993, "Database Language - SQL", American National Standard X3.135-1992, January 1993

Bukhres, O.A., Elmagarmid, A.K. (ed.), 1995, "Object-Oriented MultiBase Systems", Prentice Hall

Cattell, R.G.G., (ed.), 1996, "The Object Database Standard: ODMG-93", Release 1.2, Morgan Kaufman Publishers, San Mateo, California

Ceri, S., Pelagatti, G., 1984, "Distributed Databases: Principles and Systems", McGraw-Hill

Dogac, A., Kilic, E., Ozhan, G., Dengi, C., Kesim, N., Koksal, P., 1995, "Experiences in Using CORBA for a Multidatabase Implementation", DEXA Workshop presentation, London, 1995; Springer, LNCS 978, Berlin

Dogac, A., Dengi, C., Kilic, E., Ozhan, G., Ozcan, F., Nural, S., Evrendilek, C., Halici, U., Arpinar, B., Koksal, P., Mancuhan, S., 1996, "A Multidatabase System Implementation on CORBA", Proc. 6th Intl. Workshop on Research Issues in Data Engineering RIDE 96, IEEE Computer Society, Los Alamitos

Gardarin, G., Finance, B., Fankhauser, P., 1997, "Federating Object-Oriented and Relational Databases: The IRO-DB Experience", Proc. 2nd Intl. Conf. on Cooperative Information Systems CoopIS 97, IEEE Computer Society

Gupta, A., Mumick, I., 1996, "Materialized Views", MIT Press, Cambridge, MA

IBM, 1996, "Object Collection Service", OMG Document Number 96.5.5, IBM submission paper

Immon, W., Kelley, C., 1993, "Rdb/VMS: Developing the Data warehouse", QED Publishing Group, Boston Massachussets

Landers, T., Rosenberg, R.L., 1982, "An Overview of MULTIBASE", In: Distributed Databases, H.J. Shneider (ed.), North-Holland

Litwin, W., 1985, "An Overview of the Multidatabase System MRDSM", In: ACM Annual Conference, Denver, October 1985

Musser, D.R., Saini, A., 1996, "C++ Programming with the Standard Template Library", Addison-Wesley, Reading

OMG, 1991, "The Common Object Request Broker: Architecture and Specification", OMG Document Number 91.12.1, Revision 1.1

OMG, 1995, "CORBA Facilities Architecture Specification", OMG Document Number 97-06-15, Revision 4.0

OMG, 1997, "CORBAservices: Common Object Services Specification", OMG Document Number 97.2.24, rev. ed.

Sheth, A.P., Larson, J.A., 1990, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, vol 22, no. 3

SunSoft, 1995, "Rogue Wave Tools.h++ Class Library, Introduction and Reference Manual", SunSoft, Mountain View, USA

Widom, J., 1995, "Research problems in data warehousing", Proc. 4th Intl. Conf. on Information and Knowledge Management, pages 25-30, Nov.1995