DISTRIBUTED OBJECTS AS A LEGACY INTEGRATION MECHANISM

J Sventek

Rapporteurs: Cecília M.F. Rubira and Rogério de Lemos

DISTRIBUTED OBJECTS AS A LEGACY INTEGRATION MECHANISM

Joseph S. Sventek
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304
USA

An object model for application components permits component users to concentrate on the component interfaces without worrying about the implementations. The work described in [1] was concerned with the use of such a model for designing and constructing new systems/applications; the encapsulation behind the model also provides scope for the integration of legacy applications and data.

This paper explores various techniques for integrating legacy applications and data into object-oriented, distributed applications. Access to source for a legacy application will prove to be the gating factor for tight integration, although non-negligible integration is possible when restricted to binaries, especially if the application supports a callable API. Appropriate architectural structures in the distributed object infrastructure can enable the tight integration of legacy data, as well.

1.0 Motivation

Historically, enterprises have used monolithic application programs; these applications usually generated data in the form of files which could be further manipulated by the creating application or a (usually) small set of related applications.

Enterprises have accumulated vast amounts of data through the use of these traditional applications; this data represents a large part of the information equity of each enterprise. Since applications come and go over time, one needs expertise in a vast array of applications to be able to access all of this information; the ability to integrate this information to form synthetic data is difficult, at best, if not downright impossible.

To be sure, some enterprises have solved part of this problem by forcing all of their data into database systems, both relational and hierarchical. Report generators based on 4GLs can then be used by individuals in the enterprise to integrate different elements from the database into synthetic data elements; these synthetic elements, themselves, may be placed back into the database.

Even for enterprises which have made such a choice, the advent of personal productivity tools on workstations and personal computers has contributed to an explosion in the amount of generated data which does *not* fit into the database/4GL model. Yet, this data is part of the enterprise's intellectual equity, and there is often the desire to integrate this information with information in the databases to yield richer synthetic data.

Distributed object technologies are becoming popular mechanisms for constructing components that can be integrated into applications/compound objects tailored to the needs of a particular enterprise. As with any new computing paradigm, it is important to understand how an enterprise can bring its information legacy/equity (applications and data) forward into the new paradigm.

The remainder of the paper describes various mechanisms for using distributed object technologies to bring forward an enterprise's information legacy.

2.0 Encapsulation

The key aspect of object-oriented software technology for enabling integration is encapsulation - i.e. the interface to a component, including syntax, semantics, and protocols, completely describes a component to prospective clients of the components. Moreover, component suppliers are free to innovate behind the interface along all dimensions that are not constrained by the interface. It is this latter characteristic that eases legacy integration, since the object implementer is free to support the interface in any way possible.

There are three types of legacy applications/data that may need to be brought forward:

- applications that provide computational services; these need to be supported as computational objects in the new paradigm
- applications that generate file-based data (e.g. spreadsheet program); each spreadsheet is to be considered as an object in the new paradigm
- data that is stored in database systems; we will concentrate upon data stored in relational systems, although the discussion will carry over to hierarchical systems, as well

Prior to conversion of any applications/data to objects, it is essential that the interfaces that each component must support be defined. The language for expressing these interfaces is normally dictated by the distributed object infrastructure that animates interactions between its supported objects/components.

While some of the techniques for encapsulating applications and databases are common, it is informative to discuss them separately.

2.1 Objectifying Applications

2.1.1 Access to Source

The most straightforward way to convert a legacy application into an object is to modify the source code; of course, this assumes that the source code is at hand. From the interface specifications that the component must support and the mapping of distributed object concepts to the programming language in which the application is written, the appropriate code to support the interfaces can be inserted into the application.

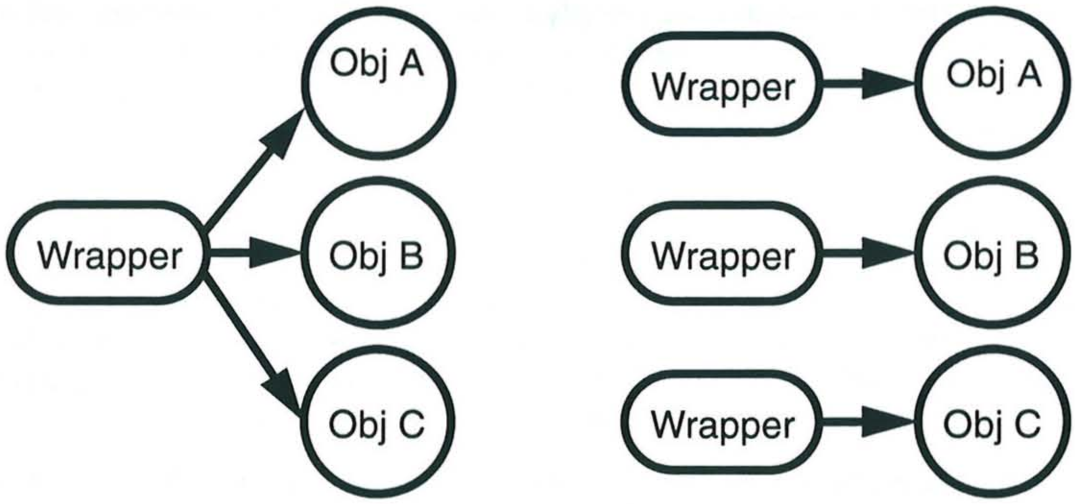
Encapsulation achieved in this way permits the resulting components to be completely integrated with other components built to the same set of integration interfaces. On the down side, this intrusive style of encapsulation requires that the programmer have a good understanding of the original code base to guarantee that the modifications do not adversely affect the original code.

2.1.2 Wrapping Binaries

In the more typical situation, one does not have the source to legacy applications. In these cases, one must construct a wrapper to the existing application which supports the component interfaces; the methods behind the interface operations then use a variety of mechanisms to induce the original application to perform requested operations.

These wrappers usually execute in an address space different from the one which animates the original application. The developer is free to design the wrapper code such that animation of the wrapper in a single address space can represent several objects/components of the same type; alternatively, it can be designed such that a separate wrapper process is needed for each component. These options are depicted graphically in Figure 1.

Figure 1: Wrapper/Legacy Object Relationships



Note that the separation of the wrapper and legacy application into different processes provides scope for the wrapper and application to be remote from one another. This wrapper technique over a distribution infrastructure is commonly being exploited to downsize from mainframes to mid-size server systems.

Typically, the legacy application only supports the functional interface of the object/component. In addition to providing a mapping of the functional interface to invocations of the legacy application, the wrapper must implement all integration policy interfaces mandated for the enterprise, as described in [1].

In general, there are two mechanisms by which the wrapper can interact with the legacy application:

- Some recent applications (e.g. Microsoft Excel) support a callable API by which other programs/applications can access the functionality of the legacy application
- The wrapper masquerades as a user by manipulating the input/output channels of the process animating the legacy application

Obviously, supporting a callable API makes the legacy application easier to encapsulate with a wrapper. Unfortunately, the number of applications accessible in this way is quite small.

2.2 Objectifying Databases

As described earlier, the intellectual equity of many enterprises is stored in database systems. Any movement to an object-oriented environment to achieve enterprise integration will require that the data contained therein be integrated into this environment.

The primary mechanism for encapsulating and integrating database systems into this environment is using the wrapper technology described in the previous section. Database systems typically provide a callable API for accessing their contents; even if a particular database system does not, most support queries in SQL and return appropriate records on an output channel that can be parsed and processed by the wrapper.

The fact that the wrapper and database engine can be in separate address spaces can be exploited in mainframe downsizing. The typical scenario is to construct the wrapper to run on a mid-size server system which is linked to a mainframe and its databases by a (typically proprietary) communications protocol. The wrapper receives operation requests for the objects in the database, translates them into queries on the database system, receives the responses, and translates/forwards the responses on to the original client.

Wrappers designed for encapsulation of database systems usually follow the leftmost relationship structure in Figure 1. The key integration dimension for the wrapper designer is the granularity of objects in the database. Two granularities come immediately to mind:

- The database is one huge object - in this case, the wrapper provides an interface to the data in the database object by supporting queries on the database object (usually in SQL). The answers to the queries are in terms of things in the database, not interface references to objects.

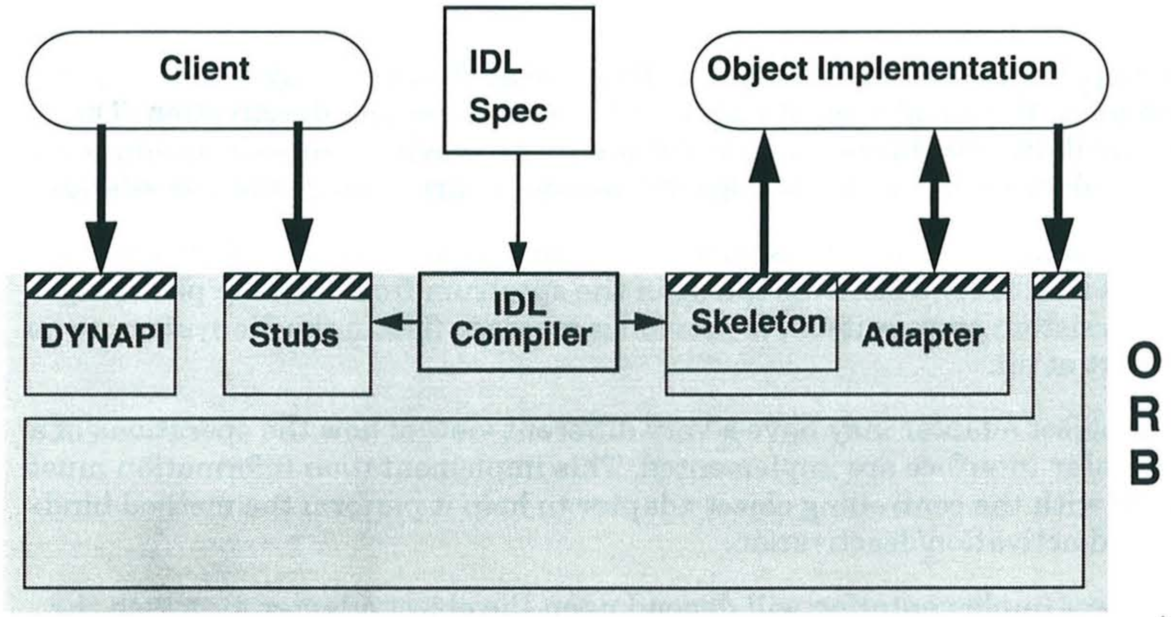
- Each row of each table is an object - in this case, each table determines the type of a class of objects. Each object in the database responds to operations to get/set the attributes of the object. An additional "database object" supports SQL queries; the answers to the queries are in terms of interface references to the matching objects.

Note that this choice is very visible to the programmer invoking operations on the database. In the coarse-grained case, the programmer is aware that he/she is interacting with a database. In the fine-grained case, the programmer can choose to be aware of the domain created by the database engine; it is not required. Manipulation of individual objects in the database is effected in the same way as any other object in the system.

3.0 Infrastructure Support

It is possible to construct the interaction infrastructure between objects such that the integration of database systems is eased. A schematic structure is shown in Figure 2. Each object implementation is constructed within the framework of an object adapter; the object adapter provides the common services necessary to construct an equivalence class of object implementations.

Figure 2: Interaction Infrastructure



3.1 Object Adapters

An object adapter is responsible for the following functions:

- generation and interpretation of object references
- method dispatch
- security of interactions

- object activation and deactivation
- facilitate access to persistent storage
- registration of implementation classes

Typically, an object adapter has its own idea of object identifiers. When a reference to one of its objects emerges into the outside world, it must be put into a form that the communications substrate knows how to handle. This does not imply that an adapter must generate such references for all of its objects - this only needs to be done when a reference escapes as an argument/result of an invocation.

All invocation requests to an object *logically* go through the object adapter. As such, the object adapter can be involved in method dispatch; some adapters may choose to handle all of method dispatch themselves, while others may relegate all of it to the skeleton.

The object adapter must be involved in the security of messages which are used to implement object interactions. If any mandatory controls have been selected, such as end-to-end encryption, then the adapter may be involved. It will most certainly be involved in enabling any required discretionary controls.

To support the illusion that objects are active at all times on system with finite resources, the adapter must handle object activation and deactivation. The responsibility for this is firmly in the adapter's province since the communications substrate has no knowledge of how objects are managed by the adapter.

An object adapter may choose to facilitate access to persistent storage by the objects that it controls. This can span the spectrum from actually providing the persistent storage itself, to providing access to files in the file system, to no support at all.

Each object adapter may have a very different view of how the operations of a particular interface are implemented. This implementation information must reside with the controlling object adapter to help it perform the method binding and activation/deactivation.

An object implementation will depend upon the object adapter for which the implementation is targeted. It will also require an IDL compiler which generates a skeleton appropriate to the targeted adapter.

3.2 Specific Object Adapters

The CORBA specification [2] only defines a single object adapter, the Basic Object Adapter; the BOA was designed with object implementations that are individually animated in POSIX processes in mind. The BOA is therefore concerned with how to activate a process when needed, how that process faults in the persistent state associated with the object, how the communication endpoints to the object's process are encoded in object references to permit clients to bind to objects and to enable location transparent access.

An object adapter for a database system, on the other hand, would be quite different. All of the objects in the database (I am assuming here the fine-grained definition of objects from Section 2.2) are animated by a single address space, which most likely would be activated at system boot time. The activities of the wrapper in the previous section actually map very nicely into a database-specific object adapter. Since object adapters are expected to have a tight relationship with other components of the infrastructure, it is expected that providing the wrapper functionality as an object adapter rather than as a separate, proxy process will be more efficient in execution.

4.0 Examples

4.1 Application Encapsulation

All of the OLE-enabled applications that are beginning to appear in the marketplace are examples of legacy integration by source modification. The OLE interfaces define an integration policy; for component applications to interwork correctly with each other, they must be modified to support the OLE interfaces.

HP demonstrated a number of encapsulated applications using the wrapper technique during the required OMG demo during the deliberations over the Object Request Broker submissions which led to CORBA. A discussion of some of these applications can be found in [3].

4.2 Database Encapsulation

4.2.1 SQL Server

Over the last few years, NASA has deployed several orbiting telescope platforms in a variety of frequency ranges. Given the investment in these orbiting platforms, and the desire to support cross frequency correlation by NASA-funded researchers, NASA has funded the development of the Astrophysical Data System (ADS).

For each orbiting platform, the data is downlinked into an archival system specific to the platform. The data is usually stored using a commercially available relational database system; unfortunately, it seems that each archival system uses a different RDMS system.

The ADS has constructed SQL servers that wrap the specific RDMS used by the different archival systems. A client application which permits the researcher to issue SQL queries on the multi-spectral data and which performs local joins of the resulting tuples knits these SQL servers together and provides value to the investigators. This simple integration of these individual legacy databases has enabled multi-spectral correlation that was impossible before the advent of the system.

4.2.2 Object Adapter

An insurance enterprise is storing claims information in a relational database; this includes elements of many media types, not just text (e.g. digitized audio of the telephone interview with the claimant). They desire the ability to integrate this information, together with data from personal productivity tools, as part of their claims processing.

HP has prototyped a database adapter in our Distributed Smalltalk ORB implementation which supports the different elements of the database as objects; we also prototyped a set of policies that enables the claims processing information integration desired by the enterprise. We are now in the process of turning the prototype adapter into a generally available product.[1]

5.0 Summary

The key aspect of distributed, object-oriented technologies, of interface conformance without imposing implementation constraints, makes these technologies perfect candidates for integrating legacy applications and data. Several mechanisms for integrating these legacies have been discussed; the most likely for most applications is wrapping the legacy application/database with code that conforms to the object model and enterprise policies. The CORBA notion of an object adapter is particularly suited for integrating database systems into the object-oriented enterprise.

6.0 References

[1] Sventek, J., "System Integration Using Distributed Objects," these proceedings.

[2] The Common Object Request Broker: Architecture and Specification, Document Number 91.12.1, Object Management Group, 492 Old Connecticut Path, Framingham, MA, 01701, 1991.

[3] Sventek, J., "The Distributed Application Architecture," Proceedings of the International Conference on Enterprise Integration Modeling Technology, Hilton Head, SC, June 1992.

1. Note that the Object Data Management Group (ODMG) is in the process of defining a standard object adapter in the CORBA scheme for accessing objects in object-oriented databases.

DISCUSSION

Rapporteurs: Cecília M.F. Rubira and Rogério de Lemos

Lecture One

During the presentation Professor Randell pointed out that when performing system integration there is usually a need for a new component to be created. On these premises Professor Randell questioned whether the two level distinction on software componentry really ought to be important and to what extent one should be able to abstract from these differences. Dr Sventek answered that there exists already a language independent model for defining the interface of components, and because of the availability of classes and frameworks in C++, for instance, instead of constructing a new component from scratch, it might be much easier to construct this component using those classes from a library already available.

After the talk Dr Aho asked for the speaker's personal experience of how much reuse people have been able to achieve by using object technology, particularly across different product lines. Dr Sventek answered that the reuse has been pretty substantial, mainly in analytical applications, such as spectrum analysis. He gave the example of a product which has undergone three evolutionary models in which from the second to third version of the product 60% of the code was taken from the library, allowing the product to be put on the market six months earlier.

Professor Rechtin made a comment that in his opinion the reuse of software during design is difficult to achieve because during design the initial assumptions concerning the component might not be available, and the number of existing errors in the component are not known, implying that the certification of the component might be much harder than just building the component from scratch. Moreover, considering the different cultures across different companies the reuse achievement becomes more unrealistic mainly in the design phase. He continued by saying that what seemed to work was when you are in the same domain, preferably in the same company with the same culture, as a consequence you can have better understanding of what might work. Dr Sventek answered that one of the possible gains that you obtain from reusing components is that you do not need to re-certify at the component level, but you still have to do the system level certification, however if the initial assumptions no longer hold then the certification no longer holds.

Dr Kramer asked which were the means provided to put the different objects together: formalisms, special integration languages or just assume that the available platform gives adequate support. Dr Sventek answered that he actually expected people like Dr Kramer to provide tools and guidelines. He also added that in terms of operational level, all the major subsystems being constructed from this technology are done by using C++ and the vertical market architecture. The latter allows one to understand how to put together the components, dictating how C++ should work.

Professor Randell made a remark that in his view the present state of object-oriented programming is alarming because of the lack of standardisation, for instance, in the terminology being used. In this context, he asked whether the situation was going to get worse before getting better. Dr Sventek answered that he had the hope that it would not. He added that there was no guarantee that different applications from different vendors would be able to interact, although the technology was available, due to political problems there was no willingness of putting different applications together.

Dr Aho asked the speaker to comment on the topics of object-oriented design and architecture. Dr Sventek answered that there is a real big gap between the business process and the actual things that you can orchestrate and implement. He continued by saying that often there were difficulties in the business process in finding an adequate granularity of reuse which satisfies the customer's requirements to reusable components.

In the end Professor Rechtin made a remark that in his opinion object-oriented design was a very natural way of reasoning about systems. He added that object-oriented design is better than the strict hierarchical model.