BROWN UNIVERSITY
Department of Computer Science
Master's Project
CS-95-M16

"Building a Client Application in the CORBA Environment Using HyperDesk's Object Services"

by

John K. Martin

Building a Client Application in the CORBA Environment Using HyperDesk's Object Services

by

John K. Martin
Department of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University.

May 1995

This research project by John Martin is accepted in its present form by the Department of Computer Science at Brown University in partial fulfillment of the requirements for the Degree of Master of Science.

Stanley B. Zdonik

Professor Stanley B. Zdonik
Advisor

Building a Client Application in the CORBA Environment Using HyperDesk's Object Services

John K. Martin
Brown University

Table of Contents

1.0 INTRODUCTION

1.1 Objective

The objective of this project was to investigate the adequacy of the object services which were provided with the HyperDesk Corporation's Distributed Object Management System from the perspective of a client application. Three main services were provided in the initial release: the Interface Repository, the Implementation Repository, and the Location services. We also hoped to identify some of the unique design and implementation issues involved in building client applications in the environment defined by the Object Management Group's CORBA standard. An object browser was chosen as the example client application to test these services because it would require much of the non-editing functionality which they provide. Such a browser would also be a necessary part of a development environment in which software engineers used preexisting objects in the management system to build new object programs. The browser was constructed using only the calls provided by these services to query and retrieve object data. Since HyperDesk's Interface Repository was built to support the operations defined in version 1.1 of the OMG's CORBA standard, it allowed us to investigate whether the interface defined by this standard, in tandem with the additional functionality provided by HyperDesk, was an adequate base upon which tools such as the object browser could be built. An attempt was made to detail the positive or negative effect of the operations defined in HyperDesk's implementation of the Interface Repository which differed from those defined in the CORBA standard. An effort was made to isolate a few small changes to the object service definitions which would increase their functionality from a client program's perspective.

1.2 Organization

This document begins with a short section on why an object browser is needed to support development in the environment defined by the OMG's Common Object Request Broker Architecture standard (CORBA). A description of the purpose of the OMG and a short technical overview of the CORBA standard and the Object Management Architecture standard (OMA) is then presented. This overview should enable an understanding of the technical issues discussed in the rest of the paper. Readers who are unfamiliar with CORBA and the HyperDesk Distributed Object Management System (HDDOMS) product may wish to consult this section first. A comparison of some of the important operations in the Interface Repository provided by HyperDesk and their counterparts in the CORBA specification is then presented with discussions on the positive and negative aspects of the differences found. Many significant issues related to the design of HyperDesk's object services were discovered when designing and building various parts of the browser. These issues are presented in section 6. Other topics relating to the construction of the user interface for the object browser are then described. The scope of this discussion is then widened to encompass some of the unique issues involved in building client applications in this environment. We conclude with a summary of the major points made in the paper and a brief discussion of some areas that were not fully explored by this research and deserve further attention in the future. A basic user guide for the browser is included as an appendix.

2.0 PURPOSE OF THE OBJECT BROWSER

The OMG is in the process of defining a set of standards which allow an application developer to access objects without regard to the machine or operating system that they run on, the language that they were written in, or the network through which they are accessed. Such an environment would clearly aid in the process of rapid application development by removing many complexities from a software development task. Programs can easily be constructed in this environment by combining preexisting objects with newly defined objects. One of the driving forces behind the OMG is to provide an environment which facilitates this reuse of code. As stated in their Object Management Architecure Guide,

> "A major goal (of the OMG) is to define a living, evolving standard with realized parts, so that applications developers can deliver their applications with off-the-shelf components for common facilities like object storage, class structure, peripheral interface, user interface, etc." [1]

The use of existing objects can simplify a developer's task by hiding complexity, but how is an engineer going to discover what potentially useful objects already exist? How can the functionality of these objects be investigated? The answer to these questions could lie in various textual resources stored in the computer's filesystem or in hardcopy specifications. Another, perhaps more user friendly solution, would be an online browser which can search for objects based on their containment structure and attribute values. A user could request that the browser display selected information about both the structure and contents of selected objects.

The CORBA standard does provide a textual language called IDL for defining objects in a manner independent of that of existing programming languages. One could presumably scan these files to discover the structure of objects, but this task could prove onerous, especially when one considers that there may be hundreds or even thousands of service objects, each possibly containing large numbers of subobjects. These IDL files could be stored in many different directories across the network, some of which a developer may not be able to access. Even if the developer can access them, it may be quite difficult to discern which objects may be potentially useful from a list of short file names. IDL files may not even exist for certain objects, as HyperDesk (and the OMG) have provided a dynamic means of defining objects which does not create IDL files. Attribute and parameter type definitions for a single interface can be scattered across multiple IDL files making a developer's job even harder.

Hardcopies of IDL files share the problems of the online files and there is no guarantee that the information in a particular hardcopy is up to date. Neither online files or hardcopy resources can provide any information on the attribute values of an object. An online browser, on the other hand, can provide up-to-date information about both the structure and content of all currently installed objects. It can provide facilities for a user to maneuver through the containment hierarchy and select objects at any level in this hierarchy. Information about attribute and parameter types can be had at the touch of a mouse button. Clearly, such a tool would be a indespensible resource to an application developer.

3.0 DESCRIPTION OF THE OMG AND THE CORBA INTERFACE STANDARD

3.1 The OMG and CORBA

The Object Management Group (OMG) is composed of a group of independent software vendors. Its mission is to develop and promote standards regarding distributed object oriented applications. The CORBA standard was developed by the OMG as the programming interface to the Object Request Browser (ORB). The ORB is the fundamental mechanism defined by the OMG which allows users to make calls to an object's operations without concerning themselves as to the actual location of the object, the implementation of the object, or how the call is made across the network to access the object. The CORBA standard was adopted in 1991 and several implementations of it are available today.

The ORB is the central component in the OMG's view of the world. It

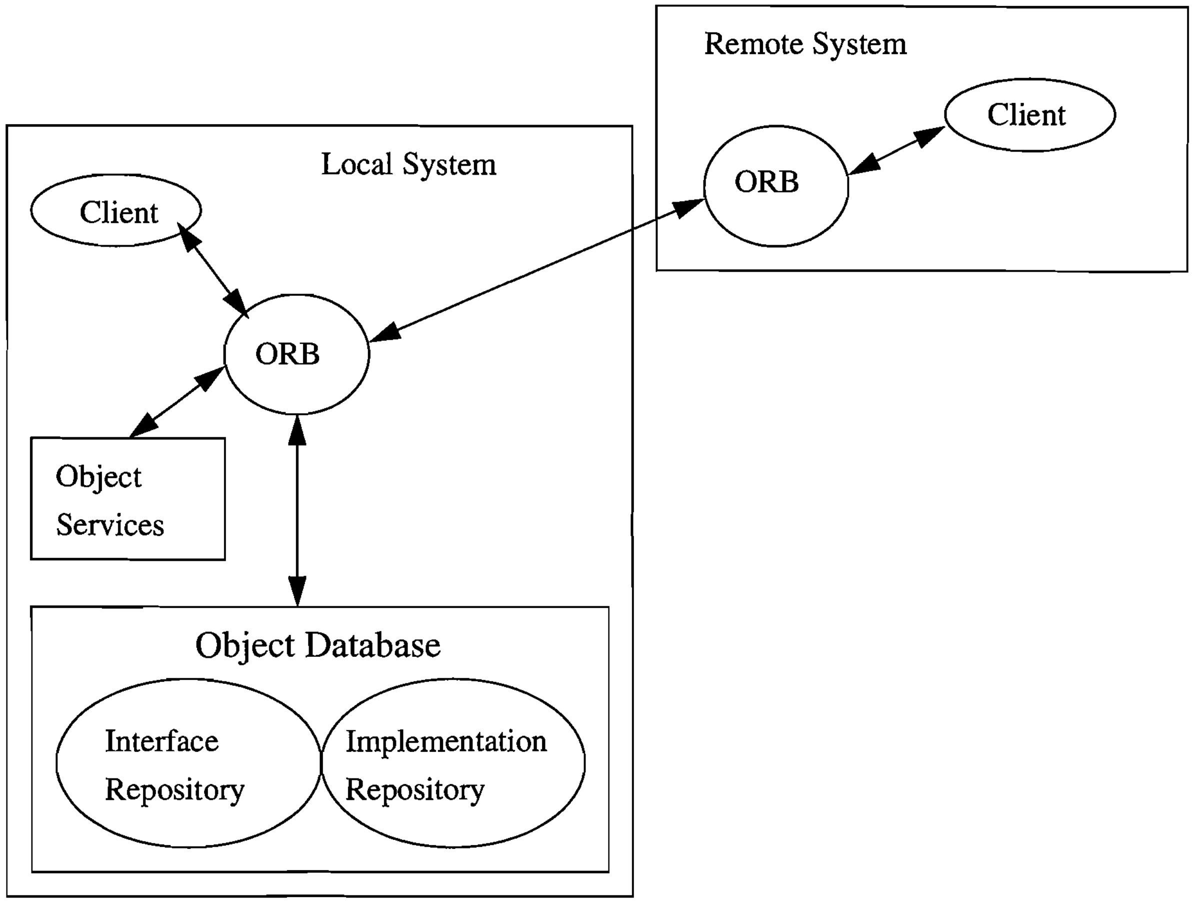


Figure 3.1: HDDOMS CORBA Implementation

handles all requests to execute an object's operations be they from a local client program or from a remote ORB. If a requested object is not held in its immediate environment, the ORB will route the request across the network to another ORB, receive the answer back and return the resulting data to the user. (See Figure 3.1) This process is hidden entirely from the client program. This hiding of internal complexity, sometimes called transparency, allows an engineer to rapidly develop client applications by returning the engineer's focus to the real problem at hand and away from any low-level intercommunication issues.

Objects in the ORB's environment can be stored in a number of places including an object oriented database, the computer's filesystem, or in a library. Objects are accessed through a software device called an object adaptor which provides mechanisms for object activation, object identifiers and management services related to an objects state. HyperDesk's implementation, HDDOMS, provides two object adapters: ODBOA and BOA. ODBOA, a proprietory adapter, stores objects in an object oriented database (Object Design's ObjectStore). It automatically provides a number of services for the objects it controls. (See Figure 3.3). It is used to store all of the objects in the Interface and Implementation Repositories. (See definition below). The BOA, defined by the OMG, is a general purpose adapter that allows objects to be stored in a variety of places. Only a few basic object services are provided with the BOA. All other services must be implemented by an object developer. It is important from the browser's perspective that all (or at least most) objects, regardless of the adapter they use, provide services to return attribute content and to move through containment.

There are two categories of utility programs defined by the OMG's Object Management Architecture (See Figure 3.2). One is called the Object Services, the other, Common Facilities. Object Services are composed of those functions which aid in the handling or use of objects. All other useful programs such as electronic mail and help utilities are lumped together under the Common Facilities category. The HyperDesk product, HDDOMS, currently provides two object services which hold information about objects, the Interface Repository (partially defined by version 1.1 of CORBA) and the Implementation Repository (named but not defined in version 1.1 of CORBA). As their names suggest, the Interface Repository holds information about an object's interface (operations, attributes, etc.) and the Implementation Repository holds information pertaining to an operation's implementation (methods, object adapters, etc.). All of the information in the repositories is provided by object interfaces which conform to the OMA's object model and have CORBA defined IDL interfaces. The two repositories are linked through the ProcedureDef object. (See Figure 3.3).

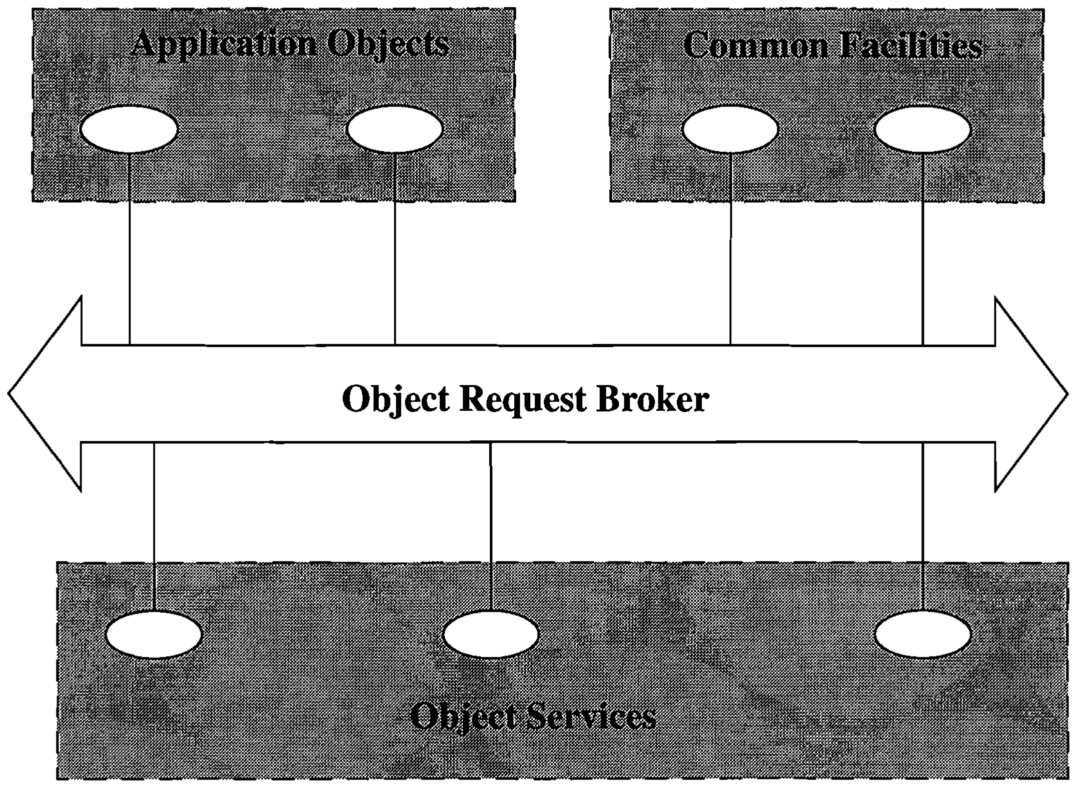


Figure 3.2: Object Management Architecture

This connection allows a user to query which machines an operation, and ultimately an object, can run on. Object Services and Common Facilities are accessed in HDDOMS through another service called the Location Service. Once a user has logged into an ORB, the Location Service can provide a list of object identifiers for services and facilities in the users view.

3.2 Object Model

It is important to present a short description of the object model used by the OMG. An object under this model is considered to be any encapsulated program which provides services to client applications. The interface to an object describes the services that an object can provide. A data type defines a legal set of values for an attribute or parameter that has that type. There are ten basic data types and five constructed data types which are built from the basic types (i.e. structures, arrays, etc.). An object operation provides a unique service and is described by a signature. This signature contains specifications for an operation's parameters, results, exceptions, and execution semantics. A single operation can have multiple methods which run under different machine types. In the future, ORB implementations will allow multiple methods for different operating system and windowing environments. The ORB automatically picks the correct method to run in a particular environment. Methods can be written as binary executables, scripts for interpreters, or program executables.

A CORBA defined object can inherit attributes and operations from one or more parent objects. An object can inherit an operation with the same name from two different parent objects if and only if the parents inherit the operation from the same source object. If the operation in the parent objects are defined in different objects, then the operation cannot be inherited. Objects which inherit from the ::Bin class (See Figure 3.1) can contain other objects. An object in the ::MemberBin class can only be contained by one parent object. Objects in the ::ChildBin class can be contained by multiple parents.

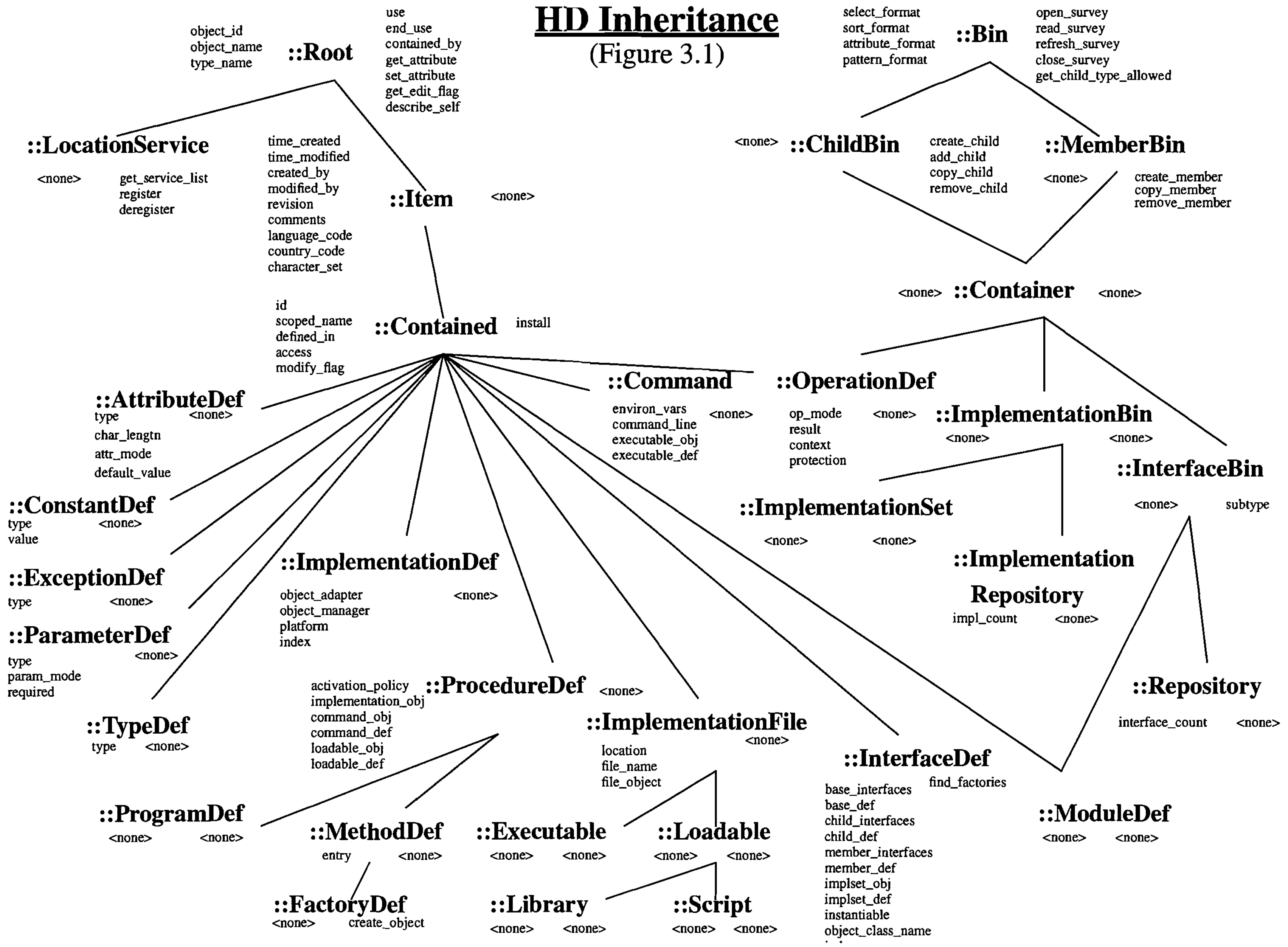
The current specification for the ORB allows only for the storage of reasonably high level objects because of the overhead required for each object. Objects such as a spreadsheet cell cannot be effectively stored under the current model. Providing a means to store and access fine grained objects under CORBA is an open area of research.

3.3 Interface Repository

The specification of the Interface Repository under the CORBA standard is admittedly incomplete. The names of certain objects in the Interface Repository

HD Inheritance

(Figure 3.1)



are defined along with the names and types of the attributes within these objects. Only a few object operations have been specified. The standard's developers did not expect that these operations would provide an adequate base upon which to build a program like the object browser. They were designed to provide basic functionality for object access.

> "The interface specifications for the Interface Repository objects define a set of basic operations for clients who want to access these interface objects. They are not intended to provide sufficient semantics for the construction of basic interface browsers or command-line interfaces to the Interface Repository, nor to provide an administrative interface." [5]

The Interface Repository objects specified by HyperDesk were designed to provide a more complete interface. The inheritance hierarchy for the objects defined in HyperDesk's Interface and Implementation Repositories is shown in Figure 3.3. The object classes are shown in bold. Attribute names are displayed to the left of the class name and operations are displayed on the right. The containment hierarchies for the Interface and Implementation Repositories can be seen in Figures 4.1 and 4.2.

It is important to examine some of the CORBA defined operations for the Interface Repository at a detailed level because they can differ significantly from the operations defined by HyperDesk. In the space below, we describe several operations in the CORBA definition which would be useful to a browser such as *contents*, *lookup_name*, *within*, *describe_interface*, and *describe_contents*. The function prototypes for all of these routines and their HyperDesk counterparts can be found in section 3.4. The *contents* operation is the fundamental containment routine which returns all child objects which are held in a specified parent object. This operation takes two parameters, restrict_type and exclude_inherited. Restrict_type allows a user to restrict the set of returned objects to a specific type.

Exclude_inherited permits the user to restrict the set of returned objects to only those which do not inherit from the given object. The *lookup_name* operation allows a user to search for an object with a given name within a certain number of containment levels of a specified object. It takes the parameters specified above along with a search_name and a levels_to_search parameter. *Within* returns a list of all objects which contain the supplied object. *Describe_interface* returns a description of all the attributes and operations defined in an interface. In CORBA, this means a description of all the AttributeDef and OperationDef objects contained by an InterfaceDef object. *Describe_contents* returns a description of all objects contained within one level of a specified object. It allows a user to restrict the set of objects returned by both type and inheritence and to specify a maximum number of objects that can be returned.

3.4 Comparison of HyperDesk and CORBA Interface Repositories

It is informative to compare some of the CORBA defined operations which are useful in the browsers context with the corresponding operations which were implemented in HDDOMS. One routine, *lookup_name*, which was defined in the CORBA standard, has not been implemented in the HyperDesk's Interface Repository. This routine looks for an object with a given name within a specified number of containment levels of the supplied object. Such a routine could be very useful in helping users who know the name, but not the location of the object for which they are looking. Other useful operations for browsing in the CORBA standard do have counterparts in HyperDesk's implementation: The *within* operation was implemented by *contained_by*. *Describe_contents* and *describe_interface* can be roughly emulated with the *open_survey* operation.

```
/* Defined in CORBA Container class */
sequence <Contained> lookup_name (
    IN  Identifier                   search_name,
    IN  long                         levels_to_search,
    IN  InterfaceName                limit_type,
    IN  boolean                      exclude_inherited
)

/* Defined in CORBA Contained class */
sequence <container> within()

/* Defined in HDDOMS Contained class */
contained_by (
    OUT ORB_SeqObject                container_list
)
```

There are some important differences between the CORBA standard's definition of *describe_contents* and *describe_interface* and their emulation through the use of HyperDesk's *open_survey* operation. The *describe_contents* operation returns a description of all attributes, operations, parameters, constants, typedefs, and exceptions defined by an object (i.e. contained by an object).. *Describe_interface* returns a description of a subset of the above object descriptions consisting of the attributes and operations defined for an interface. HyperDesk's *open_survey* operation returns a subset of the objects contained by a specified parent object. It, unlike *describe_contents* or *describe_interface*, provides the ability to both select and sort on attribute values. Pattern matching is allowed in these selections. These features are quite useful in the context of a browser. The exclude inherited parameter for the *describe_contents* operation allows a user to exclude objects which are parented by the given object. This feature is not explicitly provided by *open_survey*, but could be easily simulated with a pattern match on the base_def attribute.

```
/* Defined in the CORBA Container Class */
sequence <contained> describe_contents (
    IN  InterfaceName                limit_type,
    IN  boolean                     exclude_inherited,
    IN  long                        max_returned_objs)

/* Defined in HDDOMS Container class */
open_survey(
    IN  long                        return_data,
    IN  ORB_Scoped_Name             child_interface_name,
    IN  ORB_Format                  select_format,
    IN  ORB_Format                  sort_format,
    IN  ORB_Format                  attribute_format,
    IN  ORB_Format                  pattern_format,
    OUT long                        entries_returned,
    OUT long                        total_entries,
    OUT ORB_SurveyID                new_survey_handle
    OUT long                        num_attributes,
    OUT ORB_SurveyInfo              survey_buffer)

/* Defined in HDDOMS Root class */
describe_self(
    IN  Root_CharacteristicKind      information_type,
    OUT ORB_Survey_Info              information_list)
```

Another important difference between these operations is in the return values. The *describe_contents* operation returns a complete description of each object contained by a specified object. The *open_survey* operation allows the user to select which attribute values should be returned. However, the attributes selected must be common to all object types in the returned set. This means that a user can only see a subset of a contained object's attributes if there is more than one object type in the returned set. The *describe_contents* operation can be truly emulated by using *open_survey* in conjunction with an additional HyperDesk defined operation called *describe_self*. *Describe_self* returns either a description of the attributes or the operations defined for a specified object. It is not sufficient to simply apply this operation to an InterfaceDef object as this will return the attributes and operations of the InterfaceDef type, not the object type described by this interface. In order to view a description of the defined interface it is necessary to first do an *open_survey* operation and then call *describe_self* on each object that is returned by this call.

It is important to have a single operation which combines the above calls for two reasons. In the first place, making the necessary number of calls across the network to *describe_self* for a large interface is likely to cause delays. Secondly, this routine would receive frequent use, not only from the browser, but from other client applications which need to query the structure of an object which uses an adapter other than ODBOA. A query such as this would be made by first calling the *get_interface* operation which returns the object_id of the corresponding InterfaceDef object for a supplied object and then calling a "describe_contents" like operation on the InterfaceDef object. *Get_interface* can be called for any object, regardless of the adapter used, as it is part of the ORB_Interface. A "describe_contents" like routine need only be implemented for the object adapter which is used for the Interface and Implementation Repositories.

4.0 PROBLEMS RELATED TO MANEUVERING THROUGH CONTAINMENT AND THE SELECTION OF OBJECTS

One goal in building the object browser was to provide a user friendly means for an engineer to locate desired objects. In the OMA's environment this implies that a user must be allowed to move within the containment hierarchy and be able to select sets of objects in a specified container based on attribute values. Objects stored with the adapter being defined by the Object Database Management Group will have far more extensive search requirements, but HypeDesk did not provide such an adapter. The emphasis in this project was to provide a simple interface to a very basic set of search capabilities. Features such as set union, set intersection, and the comparison of objects at different containment levels are not provided. These types of selections could be have been built on top of the existing interface, but limitations of time prevented their implementation. Other features could not be placed on top of the object service functions specified in HyperDesk's Interface Repository. By bypassing this interface and dropping directly into the ObjectStore database some additional functionality could have been accessed, but this would have limited the use of the browser to objects implemented via the ODBOA. Hyperdesk's implementation of the Interface Repository would provide some serious limitations to anyone who wanted to build a browser with a more powerful search and selection capabilities. Some of the deficiences related to movement and selection are described in the section below.

4.1 Problems Related with Search and Selection

The first and perhaps most notable problem is that a user is not allowed to specify any selection criteria on the top level service objects. This is unfortunate because this is the point at which such selections might prove to be the most useful. For instance, a user may only wish to see those service objects which are related to a certain topics such as text editing or graphical display. Since there is no way to make such a search under the current implementation, a developer would be forced to browse every service object within his view. The lack of an object description attribute in the Interface Repository would make this search even more difficult. In an environment which could contain thousands of service objects, the lack of a sophisticated top level search facility would create a serious bottleneck for development efforts.

At first glance, the solution to this problem seems simple. The ::LocationService class could simply inhierit from the ::Container class (See figure 3.3). At closer inspection, we see that this will not work using the current

Interface Repository definition, because an object developer must specify at creation time what object types can be children of a specified type using the child_def and member_def attributes of the related ::InterfaceDef object. Since service objects with new types will be continually added to the ::LocationService this solution will not be effective. However, if one could specify a default value in the child_def attribute which allowed for all types of objects to be stored in a container, then this would provide a simple and effective solution.

Another problem arises from the structure of the Interface and Implementation Repositories and the lack of a sophisticated query mechanism that can search through more than one containment level. A developer may reasonably desire, for performance considerations, to see only those objects whose operations

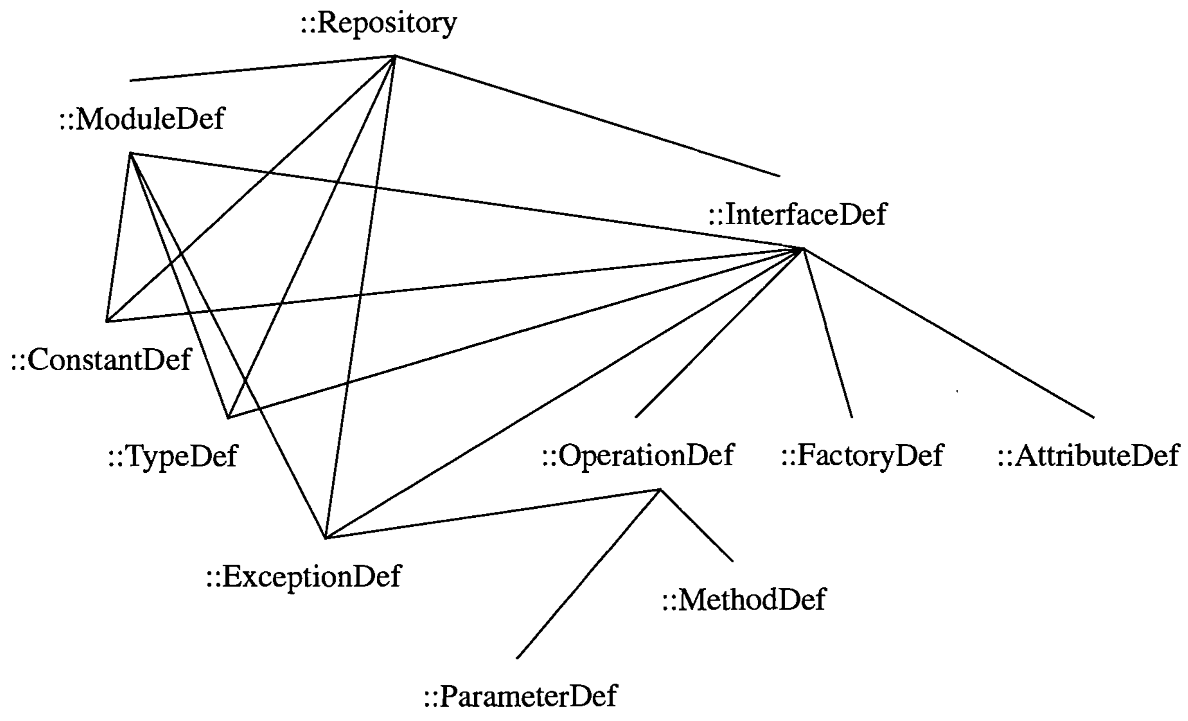


Figure 4.1: Interface Repository Containment Hierarchy

run on a particular type of hardware. This would require that the developer handcode a long and inefficient query which made sure that at least one MethodDef object within every OperationDef pointed to an ImplementationDef which has the correct platform attribute or spend hours using the browser to do the same thing. (See Figures 3.1, 4.1, 4.2) There are two ways to correct this problem. Firstly, the machines (and operating systems, etc.) upon which an object runs could be defined as attributes in the ImplementationSet or InterfaceDef objects. A default value could be created for objects which contain two operations which will not run on the same machine. The other solution is to provide a more powerful query mechanism to enable the above query. It has been suggested that the coupling between the Interface Repository and the Implementation Repository be removed. (ORBA p 12) This would make a query about which objects run under which machines impossible.[1]

The lack of a *lookup_name* routine in HyperDesk's Interface Repository makes several very useful inquires quite difficult. For instance, one may wish to see the names of the types of objects which can be children of a specified object. The name in the child_def attribute of the related InterfaceDef object may be an abstract type. We need, therefore, to discover all the object types which inhierit from this type that are not abstract types themselves. This requires finding the related object identifier for the InterfaceDef object with the given type name. With a *lookup_name* routine for scoped names, this would be easy, otherwise it would require parsing the type name and searching through each module listed in the name. A *lookup_name* routine would also be quite useful for creating online containment and inheritance charts for various service objects. (See section 7.0)

There is no standard marking for operations which return sets of objects in this implementation of the Interface Repository. Object ids can be returned in a

1. The CORBA standard does define a function called get_implemenation which is supposed to return an object in the Implementation Repository that describes the implementation of an object. HyperDesk's version of this routine returns an ImplementationDef object which describes the implementation of an operation. This does not seem to be in line with the standard or make much sense.

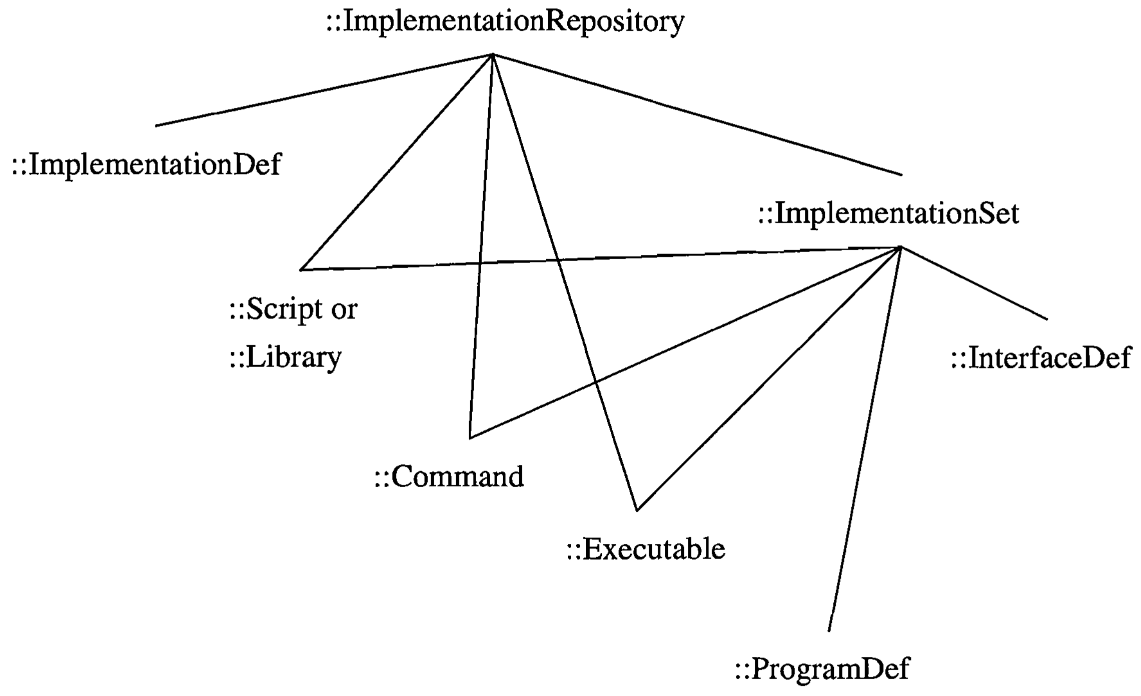


Figure 4.2: Implementation Repository Containment Hierarchy

variety of formats: a single id, an array of ids, a survey buffer containing ids, and so on. There is no way for a programs such as the browser to tell which routines have been set up by an object developer to easily access specific sets of objects. If such routines were somehow labeled and their return types standardized, the browser could allow the user to execute these operations and view the returned objects. The SeqObject datatype is the most logical return type as it can return any number of objects and is already used in the *find_factories* and *contained_by* operations in the Interface Repository. Another issue that needs to be addressed by the OMG is how can a browser at one site view the contents of the Interface Repository at a remote ORB. This is clearly an important function as the objects which a developer may wish to combine for a single client application may reside at a number of different ORBs. The ability to browse at least the Interface Repository at a foreign ORB is something that should be incorporated into the emerging inter-ORB communication standards.

4.2 The Need For Common Functionality Among All Objects In The CORBA Environment

In order for a program such as the object browser to be able to access all objects in the CORBA environment a small set of operations should be provided for all objects. These basic routines must be available in both repositories and be provided by every object adapter available. A *get_attribute* operation would be needed for all objects and a very basic open_survey routine would be needed for all container objects. These operations combined with a CORBA like *describe_interface* operation for the Interface Repository would provide a solid basis for one tool to browse all of the objects in the system. Of course, most adapters will provide a great deal of additional functionality. Additional standard functionality could be provided through levels of inheritance. For instance, editable objects would require a *set_attribute* operation, an *add_child* operation and so on. At some point the developer of an object adapter may have to chose between two or more standard levels and eventually will probably add additional proprietory functionality. The point of providing standards for different levels of functionality is that common tools will be able to act on all objects in the system, although the level of service may change with each object.

5.0 ISSUES DISCOVERED WHEN DESIGNING AND BUILDING THE OBJECT BROWSER

There are a number of additional weaknesses in HyperDesk's Interface Repository, and a few in the admittedly incomplete CORBA definition, which were exposed when trying to construct various parts of the browser. Some of these have been outlined in the sections below. In some cases we have offered potential solutions. In others the problems are related to the limitations of the chosen object model.

5.1 Displaying Attributes:

The basic types supported by the CORBA standard, (float, short, long, unsigned short, unsigned long, char, boolean, octet, and any), as well as the constructed types of string and enum can be easily displayed on a single line of a string browser. Other constructed types such as arrays, sequences, structures, and unions can be displayed on multiple lines of a string browser. Yet, people will not always view even a simple attribute type in the same way. One person may create an integer typed attribute such as age which can be displayed as an integer. Another person may use an integer typed attribute to store an encryption for the time of day which would be best viewed as a string. Still others might use the integer type to store a bit map which would be best viewed as multiple string value pairs. For another example, an attribute that is a file pointer (not defined in CORBA, but is defined in HDDOMS) may be best viewed through an editor if the file consists of text or in a graphical display if the file holds binary image data. An object implementor may desire to offer mulitiple displays of one type to adjust to the needs of various users. The only way for a generic tool like the browser to display attributes is to check the base type of the attribute and display it in its most common format. There is no current mechanism in the Interface Repository to redefine the display of the type. This follows from the view that data types in OMG's world are not objects. Object developers therefore cannot add display operations to a type object.

There is one operation that needs to be implemented for all object adapters if all object attribute values in the system are to be fully displayed by the object browser. This operation is called *get_attribute*. It enables the browser to display the content of an object's attributes. A browser could be constructed with an alternate display in which attribute values are not available for cases when a *get_attribute* function is not supplied for an object. However, since attribute values can be very useful both in understanding the design of objects and in debugging operations, it is probably best that most adapters supply an executible for *get_attribute.*

5.2 Displaying Operations:

The CORBA standard does not provide any attributes which categorize operations. As we mentioned before, this means that the browser can not identify those operations which return sets of objects. More importantly perhaps, it does not allow for such categorizations as public, protected, and private. Operations which cause permanent side effects in the environment cannot be marked. Without such tags and accompanying security, a user is free to unknowingly wreak havoc when executing untried operations. An object developer would have to be quite wary about implementing operations which were meant only to be called by other operations. HyperDesk has implemented an attribute in the OperationDef type called protection, which identifies the protection value for an operation and it provides accompaning security. If this attribute is set to PUBLIC the operation is visible to everyone. If it is set to PRIVATE, the operation is only visable to the object's creator. This attribute provides a valuable service which protects users and makes an object developer's job easier. This author recommends that the CORBA standard adopt it in its Interface Repository.

5.3 Displaying Objects

An object designer may have a standard display routine for an object type such as a personnel record which could contain a digitized photograph, an audio of the employee's speech and a variety of textual information. If a standard name for such operations was adopted they could be called by generic programs such as the object browser. Since the displays for objects could be interactive, it would be very easy to produce personalized and very powerful browsing applications for a number of different object types. The use of an interpretive scripting language such as TK-TCL could help make this process easier.

5.4 Implementation Data

The CORBA standard and the HyperDesk implementation do not provide a means of accessing any source code, makefiles, or software specifications for an operation in the Implementation Repository. This is clearly information that would be of great interest to an object developer. A versioning facility for both this information and for executable code would be a very useful and perhaps necessary tool in a development environment. A means to call an alternate executable for an operation should be designed to aid in the testing of new versions of code.

6.0 BUILDING CLIENT PROGRAMS IN THE CORBA ENVIRONMENT

The experience of building the object browser provided several insights into the development of client applications in the CORBA environment. The biggest drawback of developing applications in this environment appears to the be the substantial amount of upfront learning that is required. This is especially true, if the application requires the implementation of additional object services. The construction of a new object adapter comprises a worst case scenario in terms of the amount of new knowledge that must be assimilated.

The benefits of the environment, however, seem to far outweigh the initial cost of investment. Applications using existing object services can be rapidly developed through the use of GUI builders. Building an application with these tools mainly involves reconciling the interfaces of the widgets and the object services which are related to them This means that both the functionality and appearance of a client program can undergo radical changes with a minimum of programming effort. The rapid prototyping which this environment fosters could enable software companies to tailor their applications according to specific customer needs.

Many calls to existing object services require the use of some fairly complex data structures. They also involve quite a bit of memory allocation and deallocation through the use of some fairly cryptic routines. The speed of the rapid prototyping could be increased and the number of memory leaks reduced through the use of an interface defined in an interpretive language such as tcl. This would also open up the power and flexibility of the CORBA environment to a wider class of user.

One problem which is not encountered in more traditional programming environments is the number of ways that a service operation can fail. A request can fail because of a serious problem such as software or hardware failure or because of a more minor error such as a timeout caused by network congestion. A program must make an appropriate response for each type of error. Exception handling can take up a fairly large percentage of code which contains calls to the ORB. Object developers should pay heed to this when they design operations. Operations which logically perform a number of other operations can be quite useful to a client application developer because they can help minimize the total number of calls to the ORB and the amount of exception handling that is required. They may also be more efficient.

Another way to speed up the development process would be to create standard error handling and help facilities for the local environment. Almost every application which involves interaction with users will require these facilities. Standarized code would shorten development time and provide a similiar look and feel for a set of applications.

7.0 ISSUES INVOLVED IN CONSTRUCTING THE USER INTERFACE

One goal in designing the user interface for the browser was to allow a user to quickly access enough information about an object to use that object effectively. Another goal was to present this information in such a fashion that a user who was unfamiliar with the CORBA standard and HyperDesk's implementation could find the information he required without having to consult any outside resources. Finally, it was desired to make the interface both simple and consistent. It was not always possible to comply with these goals due to restrictions posed by the standard and the fact that the goals were sometimes not mutually satisfiable.

There appear to be three outside charts of information that a user would need to most effectively use the browser: the hierarchical graph of the base object types showing the inheritence of attributes and operations (Figure 3.1), the chart showing the containment structure of the Interface Repository (Figure 4.1), and the chart showing the containment structure of the Implementation Repository (Figure 4.2). In addition, a user might need containment charts for other service objects that he may need in a particular application. These charts can be used as road maps to guide him to the information that is required.

It would be best if such valuable resources could be recreated online. Unfortunately, this process could be quite time consuming. Creating a containment chart for a service object would require opening every module mentioned in every scoped name in the child_def and member_def attributes of every object interface in each level of the chart. Building the inheritence chart for a service object is even more complicated. It would first require building the containment chart so that all the legal object types can be listed. A process similiar to the one described above would have to be executed for each distinct object type that has not already been placed in the chart. Additional survey calls would have to be executed in order to display the operations and attributes defined by a particular interface. One problem in the efficient creation of these charts is that the base_def, child_def, and member_def attributes are a list of names and not object ids. Since under this implementation there is no way to directly convert a name to an object id each module in the scoped name must be surveyed successively to find the desired object id. If operations returning the object ids for these names were included in the InterfaceDef object this process could be sped up considerably.[1] Since scoped names are unique, another possibility would be to provide a *lookup_name* routine which returned a corresponding object id when passed a scoped name. A search facility which could provide the information shown in these charts with one call to the management system would be a very useful tool and perhaps should deserve some consideration.

In the interest of simplicity, it was decided to minimize the number of windows with which a user has to interact. Program control is centered in three windows: the object browse window, the selection window, and the attribute and operation display window. The object browse window displays the name and type of the objects in the current selection level (See Figure A.1). It is the first window that is presented to the user and it initially displays all service objects in the users view. The user is allowed to view the children of selected objects, to view the interface objects associated with selected objects, to narrow the selection of objects through pattern matching on the name and the type, and to return to the previous selection level. The browser also provides a means of reminding a user where an object actually resides. (i.e. it resides in container A which resides in

1. Note: Constructed types can not be stored as attributes under this implementation.

container B which ...)

There was a temptation to place all selection ability on the browse screen, but it was felt that this would make the window unecessarily cluttered and perhaps difficult to use. Most initial searches will probably be based on name, type, and containment. For a more complicated search involving other object attributes, the user can bring up the search window (See Figure A.2). This window initially displays all attributes which are common to all of the objects in the current selection level. The user selects an attribute, an operator, and types in a comparison value. A limited amount of type checking is done on this value when the user executes the selection to protect the user from creating a query which may crash the program. A user can specify multiple criteria for a single selection. All criteria are OR'ed together. Criteria can be AND'ed together by selecting against a new selection level. The change in the current selection level is reflected on all of the windows that the user currently has open.

The final control window displays information about an object's attributes and operations (See Figure A.3). The main difficulty in designing this window was how to present the tremendous amount of information available in a limited amount of space. An effort was made to isolate and prioritize the information about attributes and operations which a developer might need to know. This enabled us to better organize the display so that the information can be viewed in a quick and coherent fashion. The first thing a developer might like to view about an attribute, operation, or object is a descripion of its purpose and functionality. These attributes have been provided in the CORBA specification, but have not yet been included into HyperDesk's implementation. This lack of information virtually forces a developer to consult an outside resource before using an object.

A user is allowed to move back and forth through the objects in the current selection level through the use of two push buttons. The operation names and the name of the type from which the operation was inherited are automatically

Bibliography

[1] *Object Management Architecture Guide*, Object Management Group Publications, 1993.

[2] *HyperDesk Client Developer Guide*, HyperDesk Corporation, 1993.

[3] *HyperDesk Object Developer Guide*, HyperDesk Corporation, 1993.

[4] *Object Request Broker Architecture*, Object Management Group Publications, 1993.

[5] *The Common Object Request Broker Architecture and Specification*, Object Management Group Publications, 1993.

[6] *Object Services Architecture*, Object Management Group Publications, 1992.

[7] Mark A. Linton, *InterViews Reference Manual Version 3.1-Beta*, The Board of Trustees of the Leland Stanford Junior University, 1992.

[8] *ObjectViews C++ Programmers Guide*, Quest Windows Corporation, 1992

[9] *ObjectViews C++ ObjectBuild Guide*, Quest Windows Corporation, 1992

[10] "A Component Technology for the Future," Object Management Group

White Paper, Object Management Group, Inc., 1993

[11] Soley, R., Ph.D., "OMG: Creating Consensus in Object Technology," Object Management Group, 1993

[12] Probst, R., "A Software Model for the 21st Century Distributed Computing is Everywhere," OBJECT Magazine, 1993, pg. 65-67

[13] Object Analysis and Design, Vol. 1, Draft 7.0, Special Interest Group of the Object Management Group Technical Committee, 1991.

[14] Duvall, Lorraine, "The Information Needs of Software Managers: A Problem Driven Perspective", 1993 IEEE 17th Annual International Computer Software & Applications Conference, IEEE Computer Society Press, Los Alamitos, CA.

[15] Ashford, Colin, "Comparison of the OMG and ISO/CCITT Object Models", OSI/Network Management Forum, 1989.

[16] Watson, Andrew, "Object Request Broker 2.0 Extensions Interface Repository RFP", Object Management Group, 1993