



## **Object-Based Equipment Model Proof of Concept**

**SEMATECH** and the **SEMATECH logo** are registered service marks of SEMATECH, Inc.  
**International SEMATECH** and the **International SEMATECH logo** are registered service marks  
of International SEMATECH, Inc., a wholly-owned subsidiary of SEMATECH, Inc.

Product names and company names used in this publication are for identification purposes only  
and may be trademarks or service marks of their respective companies

# Object-Based Equipment Model Proof of Concept

Technology Transfer # 98113588A-TR

**SEMATECH**

*November 30, 1998*

**Abstract:** This report describes the results of a project to determine the feasibility of implementing Object-Based Equipment Model (OBEM)-compliant software components using distributed object technologies. OBEM provides software models of common semiconductor equipment and thus is useful in factory and equipment design. In general, the project team found it is feasible to adapt OBEM to the Interface Definition Language (IDL) of Object Management Group (OMG), Java, Common Object Request Broker Architecture (CORBA), and the Distributed Component Object Model (DCOM) from Microsoft. Appendixes contain source code developed by the team for OMG IDL and Java.

**Keywords:** Equipment Modeling, Object Oriented Systems

**Authors:** Gordon Davis (ObjectSpace, Inc.)

**Approvals:** Margaret Pratt, Senior Member, Technical Staff  
Dan McGowan, Technical Information Transfer Team Leader



## Table of Contents

1	EXECUTIVE SUMMARY.....	1
1.1	Mapping OBEM to OMG IDL .....	1
1.2	Implementing OBEM with Java and CORBA.....	1
1.3	Mapping OBEM to DCOM Technology .....	1
2	INTRODUCTION.....	2
3	MAPPING OBEM TO OMG IDL.....	2
3.1	Introduction.....	2
3.2	Methodology.....	3
3.2.1	Mapping Approach .....	3
3.2.2	Objects .....	3
3.2.3	Attributes.....	3
3.2.4	Services .....	4
3.2.5	Data Types .....	4
3.2.6	Notification and Reporting.....	5
3.3	Recommendations.....	6
3.3.1	Changes to the OBEM Model.....	6
3.3.2	Changes to the Documentation of the Standard.....	8
3.4	Conclusions.....	8
3.5	For More Information .....	9
4	IMPLEMENTING OBEM WITH JAVA AND CORBA.....	9
4.1	Introduction.....	9
4.2	Implementation .....	9
4.3	Creating a Java/CORBA Implementation.....	9
4.3.1	Start with IDL .....	9
4.3.2	Generate Java Interface, Stub, Skeleton, and Helper Classes.....	10
4.3.3	Implement the Functionality .....	10
4.3.4	Compiling the Source Code .....	12
4.3.5	Setting the Class Path.....	13
4.4	Running the Demo .....	13
4.4.1	Prerequisites .....	13
4.4.2	Registering the Server in the Implementation Repository .....	13
4.4.3	Starting the Server.....	14
4.4.4	Starting the Client .....	14
4.5	Conclusions.....	16
4.6	For More Information .....	16
5	MAPPING OBEM TO DCOM TECHNOLOGY.....	16
5.1	Introduction.....	16
5.2	Mapping Considerations .....	17
5.2.1	Objects .....	17
5.2.2	Attributes.....	17
5.2.3	Services .....	18
5.2.4	Data Types .....	18
5.2.5	Alarm and Event Reporting and Notification .....	18

5.3	Level of Effort .....	19
5.4	Conclusions.....	19
5.5	For More Information .....	19
APPENDIX A IDL.....		20
A.1	Complete OBEM IDL.....	20
A.1.1	Equip.idl.....	20
A.1.2	Notification.idl .....	25
A.1.3	OBEM.idl.....	28
A.1.4	SAC.idl.....	29
A.1.5	Time.idl.....	37
A.1.6	Transport.idl.....	38
A.2	Subset of IDL Used for Java Implementation .....	41
A.2.1	Equip.idl.....	41
A.2.2	Notification.idl .....	42
APPENDIX B JAVA Source Code.....		44
B.1	EquipmentResourceImplementation.java .....	44
B.2	SingleResourceDemoServer.java .....	50
B.3	EquipmentObserver.java.....	52

## **1 EXECUTIVE SUMMARY**

This report describes the results of a SEMATECH-funded project to determine the feasibility of implementing Object-Based Equipment Model (OBEM)-compliant software components using distributed object technologies, including Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM). OBEM provides software models of common semiconductor equipment and thus is useful in factory and equipment design.

A project team was formed to perform several tasks, including:

1. Map OBEM to the specific Interface Definition Language (IDL) established by the Object Management Group (OMG) standards organization.
2. Develop an Object Modeling Technique (OMT) model of OBEM using Rational Rose 4.0.
3. Develop a simple Java reference implementation of a subset of the IDL developed in Task 1.
4. Write a final report on the project's accomplishments, recommendations, issues, documentation and lessons learned.

The project's overall results are discussed below.

### **1.1 Mapping OBEM to OMG IDL**

Rather than trying to map OBEM to OMG IDL directly and literally, the team chose to approach the task from a "CORBA-centric" point of view. This was thought to make OBEM feel more natural to CORBA-oriented developers and thus enhance the possibility of its acceptance. The IDL produced by the team appears in Appendix A.

Although some difficulties were encountered, the team believes that the mapping process is viable and envisions OBEM being embodied in commercial products whose published interface is written in OMG IDL.

### **1.2 Implementing OBEM with Java and CORBA**

Using an IDL subset derived from the previously developed IDL mentioned above, the team succeeded in implementing OBEM using Java and CORBA technology. The Java code written by the team is contained in Appendix B.

The team believes that the IDL used is reasonable and that implementation of factory systems using CORBA-compliant OBEM components is feasible. The fact that the team successfully developed a sample implementation of one of the more challenging and risky portions of the IDL, together with previous experience in developing implementations of CORBA components, also leads to this conclusion.

### **1.3 Mapping OBEM to DCOM Technology**

DCOM cannot be ignored because it is a Microsoft standard. DCOM views objects similarly to CORBA, mainly in that both separate interface from implementation. DCOM's interface is defined by Microsoft IDL (CORBA uses OMG IDL) and often differs from CORBA in terminology and paradigm. DCOM primarily supports development in C++ and Java, while

CORBA language bindings are defined for C, C++, Java, Ada, Cobol, and Smalltalk. DCOM uses the Component Object Model (COM) as its object model.

Based on their success in developing OMG IDL for OBEM (a process that took about three weeks), team members believe that Microsoft IDL for OBEM could be developed in about the same time. Although actual mapping of OBEM to COM/DCOM was not done in this project, the team believes such activity is feasible.

## 2 INTRODUCTION

This report was written to satisfy the requirements of the Object-Based Equipment Model (OBEM) Proof of Concept project funded by SEMATECH. As described in the *Standard for the Object-Based Equipment Model* (SEMI Draft Document #2748), OBEM provides a software model of “the common types of physical and logical objects of which equipment is typically composed, including the equipment itself.” Specifically, the type of equipment modeled is equipment used in the manufacture of semiconductor devices. The purpose of this project was to determine the feasibility of the implementation of OBEM-compliant components using distributed object technologies such as Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM).

The tasks required in this project were as follows:

1. Mapping OBEM to Object Management Group (OMG) Interface Definition Language (IDL).<sup>1</sup>
2. Developing an Object Modeling Technique (OMT) model of OBEM using Rational Rose 4.0.
3. Developing a simple Java reference implementation of a subset of the IDL developed in task 1.
4. Writing a final report to include the following:
  - Discussion of the activities above
  - IDL developed in task 1
  - List of recommendations for the OBEM Standard to facilitate mapping it to IDL
  - IDL and Java source for the implementation developed in task 3
  - Documentation of the implementation developed in task 3
  - Discussion of issues related to implementing OBEM on DCOM
  - Lessons learned

## 3 MAPPING OBEM TO OMG IDL

### 3.1 Introduction

This section describes the approach taken to map OBEM to OMG IDL and presents recommendations for making it easier to map OBEM to IDL. Appendix A contains the IDL produced in this project.

---

<sup>1</sup> The project SOW refers to OMG IDL as “CORBA IDL,” but “OMG IDL” is the proper term according to OMG, the standards organization responsible for both CORBA and OMG IDL.

## 3.2 Methodology

Project participants began by familiarizing themselves with the OBEM 12 standard and the SEMI E39-0697 Object Services Standard (OSS) to which it refers. OBEM 13 and other applicable SEMI standards were reviewed along the way. The project's IDL is based primarily on the OBEM 12 standard, but is influenced by an early draft of OBEM 13, mostly in the area of the SensorActuatorDevice. After developing the IDL and an object relationship model, the IDL was subjected to an internal ObjectSpace, Inc. review process.

### 3.2.1 Mapping Approach

Rather than map OBEM to OMG IDL in a direct and literal manner, it was decided to approach the mapping from a "CORBA-centric" point of view. This was thought to make OBEM feel more natural to CORBA-oriented developers and thus enhance the possibility of its acceptance.

### 3.2.2 Objects

Since an IDL interface represents roughly the same notion as an OBEM object—that is, an entity that represents both state and behavior—OBEM objects were mapped to IDL interfaces. However, there is a small difference that should be noted: OBEM specifies ObjType (object type) and ObjID (object ID) attributes for each object, whereas in CORBA the object ID and object type are inherent in the object and are not formal attributes of the object.

### 3.2.3 Attributes

At first glance, it would seem that OBEM attributes should be translated to CORBA attributes: the OBEM read-only (RO) attributes would be translated to CORBA readonly attributes, and the OBEM read-write (RW) attributes would be translated to CORBA attributes (without the readonly prefix). This would be acceptable under the following conditions:

1. All attributes are required.
2. All attributes are either always read-only or always read-write.
3. In the case of read-write attributes, all values of the specified type are acceptable.

However, in OBEM some attributes are optional. A common way to handle this in IDL is to specify an exception stating that the attribute is not supported. This way, the exception can be raised and caught if the attribute is not supported by the interface. Raising exceptions is not supported in the case of attributes.

Additionally, a particular attribute in OBEM may be read-write or read-only, depending on some condition. For this, it is desirable to specify an exception that could be raised in case a client tries to set the value when setting a value is not supported.

Finally, there appear to be OBEM attributes whose values are limited to a proper subset of all possible values of their declared type. In this case, too, it is desirable to define an exception.

For these reasons, the project chose to model all OBEM attributes as one or more IDL operations. In the case of a read-only required attribute, only one operation is needed. For example, the attribute Manufacturer in the EquipmentModule object maps to a single operation string getManufacturer(); in the Equipment::EquipmentModule interface in IDL.

If attributes are not read-only or optionally read-only, however, a setting operation is required. For example, the attribute ModuleName in the EquipmentModule object maps to two operations in

OMG IDL: `string getModuleName();` and `void setModuleName(in string name)` raises (OBEM::StringParameterTooLong);.

In the case of optional attributes or optionally writeable attributes, three operations are generally required. One of the operations allows the value to be read, one allows it to be set, and one checks to see if the attribute is supported or if changing its value is supported. For example, the Units attribute in the ScalarSensor object, which can be either read-only or read-write, maps to three operations in OMG IDL: `boolean supportsChangingUnits();`, `string getUnits();`, and `void setUnits(in string units)` raises (InvalidUnits, OBEM::OptionalOperationNotSupported);.

### 3.2.4 Services

While OBEM services translate fairly directly to CORBA, there are some complications. For example, the concepts of optional services and optional service parameters do not translate easily into OMG IDL. In this implementation, optional services are handled by providing a boolean-returning operation that indicates whether the service is supported, as well as an operation corresponding to the service with OBEM::OptionalOperationNotSupported specified as one of the exceptions that could be raised by the operation.

For example, IDL interface Equipment::EquipmentResource (the EquipmentResource interface of the Equipment module) contains a boolean-returning operation `supportsAddSensorActuatorDevice` that indicates whether the interface supports adding and removing SensorActuatorDevices. The interface also contains the operations `addSensorActuatorDevice` and `removeSensorActuatorDevice`, and both of these operations support the raising of OBEM::OptionalOperationNotSupported.

While there are no examples of optional service parameters in OBEM 12, it is reasonable to consider the possibility, since the OBEM standard follows OSS conventions closely, and OSS supports optional service parameters. In the case of optional service parameters, one would probably translate them as operation parameters (since optional parameters are not supported in OMG IDL) along with the other service parameters. Designated null values (specific to each type or parameter) would be used to indicate the lack of use of a particular operation parameter on a particular call. Necessary restrictions could be enforced with the use of exceptions.

### 3.2.5 Data Types

#### 3.2.5.1 Primitive Types: Boolean, Float, Integer, Unsigned Integer, Text

These OBEM types are translated easily into IDL. OBEM Boolean translates directly to IDL boolean. The project usually translated Integer to long since there is rarely an indication of the values' magnitudes. Where it is clear that short or octet could handle the value, those should be used. While Integer could have been translated to long long to handle the largest value possible, this was not done because 64-bit integers are rarely used or needed today. A long (32-bit) accommodates values of plus or minus 2 billion, generally sufficient for an integer value. Unsigned integer was mapped to unsigned long. Floating point was translated to double since there is no indication of the magnitude or accuracy required. Again, long double was not used due to lack of precedent and the fact that such precision and magnitude generally are not needed. Text was mapped to string.

#### 3.2.5.2 Numeric

The type Numeric has no equivalent in IDL, thus presenting a problem. There are two basic approaches for solving this problem: 1) create a union of all numeric types (there are

approximately 10 in CORBA 2.2) or a subset of these, and 2) use any. While it is ideal to avoid the use of case statements, it is not really possible in this case unless you know which specific type you expect the value to be in a particular case. It is better to use union rather than any because the former provides type checking. It was decided to map OBEM Numeric to a union that includes only two types, long and double.

### **3.2.5.3 More Complex Types: Formatted Text, Numeric Text, Structure, List, Enumerated**

OBEM Formatted Text was mapped to string. Although an IDL interface or struct could be designed for this purpose, a string with special formatting characters defined should work equally well to represent simple Formatted text. Project members believed this would suffice and so did not include in the scope of this project an investigation of the text formatting design space. Numeric text was also mapped to string. All Numeric text fields in OBEM 12 are RO, although if some had been RW, it would have been necessary to add exceptions to the services used to signal a poorly formed numeric text string. Structure types were mapped to struct. Both List and Ordered list were mapped to sequence. Finally, Enumerated was mapped to enum.

Prefixes were added to the enum constants since CORBA does not allow enum constants that are identical to other identifiers, and there are several potential clashes between enum constants, module names, interface names, etc. For example, in interface EquipmentModule, there is an enum whose enumerated constants (without prefixes) are Process, Measurement, Transport, and Storage. In Equipment, which is a subclass or specialization of EquipmentModule, a reference is made to the Transport module. The OrbixWeb IDL compiler flags this as an error, because it believes Transport already is being used as an enumeration constant in the superclass interface.

### **3.2.6 Notification and Reporting**

The project believed that the existence of Alarms and Events in EquipmentResource, ReportRate in Sensor, and the descriptions of Possible Units and Nominal Value in Scalar Sensor implies a notification and reporting mechanism. It was decided to flesh this out and provide IDL for a basic notification and reporting mechanism (see modules Notification and SensorActuatorController). This was done in a way that facilitates the use of the CORBA Event Service to implement the mechanism, should such be desired.

EquipmentResource was defined to have both AlarmGenerator and EventGenerator as superclasses. These two superclasses provide services that allow other objects to find out which alarms or events an AlarmGenerator or EventGenerator can signal. Also, they allow for certain modifications of alarms and events, permit disabling and enabling alarm and event notification, and provide alarm and event subscription services.

Similarly, Sensor was subclassed from SensorReportGenerator. SensorReportGenerator is similar to AlarmGenerator and EventGenerator.

### **3.3 Recommendations**

The project's specific recommendations fall into two primary categories:

1. Areas where a change to the OBEM model might facilitate the mapping of OBEM to OMG IDL
2. Areas where a change to the model's documentation might facilitate such a mapping

#### **3.3.1 Changes to the OBEM Model**

The more the OBEM object model conforms to the CORBA object metamodel (the CORBA way of doing things), the easier it will be for IDL developers to develop IDL for OBEM. The following subsections include specific ideas for changing the model.

##### **3.3.1.1 Removal of Object Identity Attributes**

Remove the ObjType (object type) and ObjID (object ID) attributes from the objects. These are confusing to a CORBA developer because in CORBA, the object identity and type are inherent to the object.

##### **3.3.1.2 Structure Definitions**

The OBEM Structure and the IDL struct appear to be semantically different, despite the similarity of terms. The OBEM Structure has a mutability annotation (e.g., RO or RW) in each field, while the IDL struct does not. Since all structs in CORBA are passed by value, changing the value of a field in a struct does not affect a remote object. Because of the similarity in terms, an IDL designer would tend to map an OBEM Structure to an IDL struct, and this difference could create confusion, especially when a Structure such as Dataltem is labeled RO, and one of its fields is marked RO or RW.

##### **3.3.1.3 Optional Elements**

Eliminate optional services and attributes (including optionally writeable attributes). The concept of optional elements is foreign to CORBA.

##### **3.3.1.4 Data Types**

Use CORBA-compatible data types. Using CORBA-compatible data types in OBEM certainly would make mapping easier. As mentioned in the Methodology section, one of the most difficult OBEM data types to handle is Numeric.

##### **3.3.1.5 Reporting Protocol**

Define the reporting protocol. For objects that report information on a periodic and/or an exceptional basis, such as Sensor, the protocol for subscribing to the reports and the manner in which the notification is made should be specified in the OBEM standard.

##### **3.3.1.6 Exceptional Conditions**

Define exceptional conditions, such as invalid parameter values. Additionally, the information sent along with the exception should also be specified.

##### **3.3.1.7 CORBAservices**

Align OBEM with CORBAservices where appropriate. In other words, "Do things the CORBA way." CORBA-oriented developers are familiar with the CORBAservices way of propagating

events, finding objects, and so on. Therefore, defining OBEM so that it is familiar to CORBA-oriented developers and so allowing these auxiliary aspects to be mapped to CORBA services would facilitate mapping OBEM to OMG IDL. For example, the inherent event notification and reporting found in EquipmentResource and Sensor is mapped in a way that would facilitate the use of the CORBA Event Service.

### **3.3.1.8 Attributes and Relationships**

In conformance with current object-oriented practice, relationships between objects should use object references, not string identifiers. Neither should they be considered attributes. For example, the attribute table for CarrierPort lists attributes LocAssigned and LocAvailable, which are of types Text and List of Text, respectively. In the project's opinion, these should be left out of attribute tables and instead shown in object relationship diagrams with additional notes (if necessary) to clarify relationships.

### **3.3.1.9 Events**

Define specific events where appropriate. Define the protocol for subscribing to events and the manner in which the notification should be made.

### **3.3.1.10 Service Signatures**

Align service signatures with Common CORBA usage. OSS conventions specify messages as either "request" or "notification" and message parameters as either mandatory or conditional. OSS also allows message responses that contain multiple parameters. CORBA does not distinguish between request and notification, because the IDL does not specify the client or server; it assumes that any client can talk to any interface for which it can obtain a reference. Common CORBA usage returns either no value or a single value. Although inout and out parameters are allowed, they are not commonly used in IDL published by OMG.

### **3.3.1.11 Clients**

Do not assume a single, distinguished client for an object interface unless no other scenario is possible.

### **3.3.1.12 Conceptual versus Official Objects**

Distinguish between objects included on domain-oriented conceptual model diagrams and true OBEM objects for which the standard specifies their features.

### **3.3.1.13 Object Graph Navigability**

Include more explicit object graph navigability information. A simple theory of object graph navigability, though not strictly formal, illustrates the project's concerns. In an object graph, the nodes are objects and the edges are object relationships. Object graph navigability, then, refers to the existence or lack of *navigable paths* between objects, especially adjacent ones. There is a navigable path from object A to object B if object A provides a service whose purpose is to return a reference to object B.

Because the OBEM standard uses the conventions of OSS, a navigable path was assumed from an aggregate (as denoted by the open diamond) to its components. However, other types of relationships are denoted on object diagrams in which it is not known if there are one or two navigable paths (the latter occurring when objects can return references to each other). In many

cases, there should be at least one navigable path, and these navigability relationships should be specified so that the OMG IDL designer does not have to guess at the relationships.

For example, in the OBEM 12 model, it is not clear whether a CarrierPort should return a reference to a PortLocation, whether a PortLocation should return a reference to a CarrierPort, or whether each should return a reference to the other. Typically, arrowheads are used to denote navigability, and relationships that are navigable in both directions are often shown without arrowheads. If this is the case, the convention should be clarified in explanatory notes.

### **3.3.2 Changes to the Documentation of the Standard**

Where applicable, consistent documentation following common conventions is important, so that IDL developers can quickly update their understanding of the OBEM model. The following subsections include specific suggestions.

#### **3.3.2.1 Service Definitions**

Include complete service definitions. A CORBA operation is defined by the combination of operation name, return type, and parameter list. A parameter list consists of zero or more parameters. For each parameter, direction (in<sup>2</sup>, out, or inout), type (e.g., SensorActuatorDevice or double), and name (e.g., device) are specified. Ensuring that all of this information is included in service definitions will facilitate the mapping of OBEM services to IDL.

#### **3.3.2.2 Consistency of Information**

Ensure that all information is consistent. Since a CORBA specification is formal, it is important that the information be consistent. Sometimes inconsistent information has enough information redundancy that the proper information can be deduced, but this is not always true. In any case, this can be frustrating and cause inefficiency in mapping, so be sure to check both textual, tabular, and graphical information.

#### **3.3.2.3 Usage Information and Examples**

Provide usage information and examples. Since creating a useful interface requires both an understanding of potential implementation models and probable usage models, including this information will facilitate mapping the services to IDL. The usage information is especially important, and it is helpful to give the purpose of the service, the types of collaborators or clients you would expect to use it, and the way in which you expect it to be used. Include one or more examples, if possible.

### **3.4 Conclusions**

Although the project team had some difficulties in mapping OBEM to IDL as discussed, it believes that the mapping process is viable, and envisions OBEM being embodied in commercial products whose published interface is written in OMG IDL.

---

<sup>2</sup> In CORBA, an `in` parameter transmits information from a calling process to a CORBA operation to be used in the execution of its implementation; an `Out` parameter transmits information from a CORBA operation to a calling process as the result of the execution of its implementation, and an `inout` parameter both transmits information to a CORBA operation before execution and transmits information back to the calling process after execution.

### 3.5 For More Information

For information on OBEM, see SEMI Draft Document #2748, *Standard for the Object-Based Equipment Model (OBEM)*. For information on CORBA, see *CORBA Fundamentals and Programming*, Jon Siegel et al. (New York: John Wiley & Sons, Inc., 1996) and *The Common Object Request Broker: Architecture and Specification* version 2.2, available from OMG.<sup>3</sup>

## 4 IMPLEMENTING OBEM WITH JAVA AND CORBA

### 4.1 Introduction

This section describes the approach taken to implement OBEM using Java and CORBA technology. It also provides information on running the demo. The Java code written by the project team can be found in Appendix B.

### 4.2 Implementation

The programs developed for this project form an active server<sup>4</sup> that implements a single, persistent<sup>5</sup> object that is similar to an OBEM Equipment Resource<sup>6</sup> and a single client that exercises a portion of the Equipment Resource's capabilities. This client has a console-type interface and follows a set course of action. Specifically, it does the following:

- Obtains and displays some of the attributes of the Equipment Resource
- Demonstrates that it can change the value of a read/write attribute in the Equipment Resource
- Obtains a list of events
- Displays the list to the user
- Allows the user to choose one
- Subscribes to that event
- Displays a message whenever an event notification has been received

### 4.3 Creating a Java/CORBA Implementation

#### 4.3.1 Start with IDL

To create a server using Java and CORBA, the developer must have IDL describing the public interfaces of the different objects. The project team pared down the OBEM IDL developed for Milestone 1 to produce the "Subset of IDL for Java Implementation" in Appendix A. The following interfaces remained after this removal: EquipmentResource, EventSubscriber, and

---

<sup>3</sup> This document is more commonly referred to as the CORBA/IIOP 2.2 Specification. The document can be obtained both in .ps and .pdf format at <http://www.omg.org/corba/cichpter.htm>.

<sup>4</sup> An active server is one that can send messages to its client or clients. Although a server typically can handle multiple clients, this server is only designed to handle one client at a time.

<sup>5</sup> It is persistent in the sense that it exists for the lifetime of the server, not in the sense that its state persists beyond the lifetime of the server process.

<sup>6</sup> For the purpose of this report, this object will be called an Equipment Resource, even though it does not have all the capabilities of an OBEM Equipment Resource. Specifically, it has no relationships to other Equipment Resources, to Material Locations, or to Sensor Actuator Devices. Its availability cannot be changed and it cannot generate alarms, although it can generate events.

EventGenerator. EquipmentResource is derived from EventGenerator, and EventSubscriber is the interface that should be implemented by a client.

### 4.3.2 Generate Java Interface, Stub, Skeleton, and Helper Classes

The team used the product OrbixWeb 3.0 by Iona Systems to provide the ORB functionality and generate the necessary classes, and utilized the OrbixWeb IDL compiler to compile each IDL file. For the EquipmentResource interface, the following Java classes and interfaces were generated:

- EquipmentResource: This interface is the client's view of an Equipment Resource.
- EquipmentResourceStub: This is the class of the EquipmentResource proxy on the client side. It forwards messages sent to the proxy to the proper object on the server. The developer does not need to be concerned with this class.
- \_EquipmentResourceSkeleton: This is an abstract class that dispatches incoming requests to the server.
- \_EquipmentResourceImplBase: This is an abstract class and is a subclass of \_EquipmentResourceSkeleton. This class claims to implement the EquipmentResource interface (see above) but does not actually do so (it can do this because it is an abstract class). EquipmentResourceImplementation, the class the team wrote to provide an implementation of the EquipmentResource interface, is a subclass of this class.
- EquipmentResourceHelper: This class provides utility methods for both the client and the server sides.
- EquipmentResourceHolder: This class enables the passing of EquipmentResource objects as input/output (inout) or output (out) parameters in operation invocations.

The OrbixWeb IDL compiler also generates other classes for an interface (such as \_tie\_EquipmentResource and \_EquipmentResourceOperations), but these are not used in the style of implementation the team employed. Similarly, the same series of classes are generated for EventGenerator and EventSubscriber. Classes also are generated for structs, exceptions, and other IDL elements.

### 4.3.3 Implement the Functionality

The following three classes were implemented for the demonstration described in Section 4.2, Implementation:

- EquipmentResourceImplementation provides the actual implementation of the EquipmentResource interface.
- SingleResourceDemoServer uses EquipmentResourceImplementation to make a single instance of EquipmentResource available for client interaction.
- EquipmentObserver implements the EventSubscriber interface and the course of action of the demonstration.

#### 4.3.3.1 Equipment Resource Implementation

EquipmentResourceImplementation is defined in EquipmentResourceImplementation.java. It is a subclass of \_EquipmentResourceImplBase and implements the EquipmentResource Java interface.

#### 4.3.3.1.1.1 EquipmentResource Interface Proper

The implementation of the EquipmentResource interface proper is quite straightforward, and each attribute of EquipmentResource is stored in an instance variable. The accessor and mutator operations in the interface (getDescription, setDescription, getFunction, getSerialNumber, isAvailable) are implemented in a straightforward manner.

#### 4.3.3.1.1.2 EventGenerator Interface

The implementation of the EventGenerator interface consists of getEvents, getEvent, subscribeToEvent, and unsubscribeFromEvent. Events are stored in an array held in an instance variable named events. Method getEvents simply returns the contents of the instance variable, while getEvent returns a specific Event by traversing the array looking for an Event with the desired event ID.

Subscription information is stored in a Hashtable named subscribers. This Hashtable maps each Event to a Vector of EventSubscriber instances. EventSubscriber instances are defined as instances that implement the EventSubscriber interface. Each of these objects is actually an \_EventSubscriberStub, a proxy to a remote (client) EventSubscriber.

When the subscribeToEvent or unsubscribeFromEvent method is invoked, a search must be made for the specified subscriber in the subscriber list for the specified Event. This requires comparing two EventSubscriber instances. However, since they are proxies, they may not be the same local object, even if they refer to the same EventSubscriber object in the client. Actually, CORBA defines a method on all CORBA objects called \_is\_equivalent. This method allows the comparison of two remote object proxies to determine if they refer to the same remote object. This method is used here to compare remote subscriber instances in the subscriber list.

#### 4.3.3.1.1.3 Constructors

Three constructors are defined for the class:

1. A default constructor that takes no arguments
2. A constructor with parameters to specify all EquipmentResource attributes and all Event instances that the EquipmentResource should be able to broadcast
3. A constructor that allows specification of values for the EquipmentResource attributes, all Event instances, a specific Event to be broadcast at regular intervals, and the interval of time to wait between successive broadcasts. This is specially defined to facilitate the demonstration, and spawns a separate thread to broadcast the specified event to all subscribers of the event at the specified interval.

#### 4.3.3.1.1.4 Broadcasting

The broadcastEvent method iterates through the subscriber list of the specified Event and sends the EventNotification to each subscriber instance in the list via the push method. Since push is defined in the IDL as taking an any parameter, special work must be done by the developer both on the sending and receiving sides to package the parameter (EventNotification) for sending, then provide for it to be unwrapped once it is received.

#### 4.3.3.2 SingleResourceDemoServer

SingleResourceDemoServer is defined in SingleResourceDemoServer.java. This class is not instantiated; it contains only static methods.

After validating and massaging the command-line arguments, the server static main method, containing the main server setup and processing logic, initializes the ORB. It then creates a single instance of `EquipmentResourceImplementation`. The constructor of `_EquipmentResourceImplBase`, the superclass of `EquipmentResourceImplementation`, which implicitly is called when the constructor of `EquipmentResourceImplementation` is called, connects the object to the ORB. The constructor used in this example also starts the broadcasting of the message selected in the command line arguments at the interval specified in the command line arguments. Finally, `impl_is_ready` is sent to the BOA (Basic Object Adapter). This sets up the message-processing loop for incoming CORBA messages.

#### 4.3.3.3 EquipmentObserver

`EquipmentObserver` is defined in `EquipmentObserver.java` and is a subclass of `_EventSubscriberImplBase`. It implements the `EventSubscriber` interface. Although this class provides a static main method that drives the demo from the client side, the class also is instantiated to provide a receiver for event notifications.

After processing the command-line argument, the static main method initializes the ORB. Then it binds to the `EquipmentResourceImplementation` in the server ORB with the aid of the `bind` method of `EquipmentResourceHelper`.

The result of the `bind` is a proxy to the `EquipmentResource` on the server. The actual type of this proxy is `_EquipmentResourceStub`. However, since it implements the `EquipmentResource` interface, the client considers it as an `EquipmentResource`. From this point on, the client talks to the `EquipmentResource` by sending messages to the proxy.

After the `bind`, the main method exercises the `EquipmentResource` attribute accessors and mutator by sending the proper messages to the proxy. At that point, it gets a list of all events (by sending `getEvents` to the proxy) and displays them. It then asks the user to choose one, after which the method sends the `subscribe` message to the proxy and the client is subscribed (via an instance of `EquipmentObserver`) to the selected event. Specifically, the main method creates an instance of `EquipmentObserver`, which implements the `EventSubscriber` interface, and passes this as the subscriber parameter in the `subscribe` message. If the event subscribed to is the same as the one being broadcast<sup>7</sup>, the `EquipmentObserver` instance will receive the notifications and print out what it has received to the console.

After subscribing, the main method sends `processEvents` to the BOA. This has the effect of blocking the main thread while allowing the client to receive event notification messages, which occurs by the `EquipmentObserver` instance receiving the push message from the server. This takes place via the `EventSubscriber` proxy on the server, since `EquipmentObserver` implements `EventSubscriber`, which is how the server sees the client object.

The other significant method is the `push` instance method. This method, when called, prints out a message on the client console indicating the ID and text of the `EventNotification` received.

#### 4.3.4 Compiling the Source Code

All of the source code, including code generated by the OrbixWeb IDL compiler and the three source files written by the team, must be compiled. The team used the `javac` compiler distributed

---

<sup>7</sup> Recall that the server is continuously broadcasting one particular event.

in Sun Microsystems' JDK 1.1.6 for this task. The code must be placed in a directory hierarchy according to package, which is standard Java practice.

#### 4.3.5 Setting the Class Path

The class path must be set so that the Java class files can be found by the Java virtual machine. This is done by setting the CLASSPATH variable in the environment.

#### 4.4 Running the Demo

Although both Java and CORBA are available on many different platforms, the demonstration code developed for this project was created with OrbixWeb 3.0 development tools on Windows NT 4.0. It has been tested only on Windows NT 4.0 using that ORB.

##### 4.4.1 Prerequisites

- Windows NT 4.0 (Windows 95 may work, but has not been tested)
- Iona OrbixWeb 3.0
- Compiled Java .class files accessible via the CLASSPATH environment variable

##### 4.4.2 Registering the Server in the Implementation Repository

To register the server, type the following at a Command Prompt:

```
putit -java EquipSrv Server.SingleResourceDemoServer
```

To confirm that the server is properly registered, type the following:

```
catit EquipSrv
```

The response should be similar to the following:

```
[New Connection (gdavis,IT_daemon, *,gdavis,pid=-889228743) ]
```

```
Details for server : EquipSrv
```

```
Comms. Protocol      : tcp
Code                 : xdr
Activation Mode      : shared
Owner                 : gdavis
Launch                : ;
Invoke                : ;
```

```
Marker Launch Command
```

```
*          ###ORBIXWEB### Server.SingleResourceDemoServer
```

### 4.4.3 Starting the Server

The syntax for running the demo server is as follows:

```
java Server.SingleResourceDemoServer <index of desired event> <interval between broadcasts >
    <index of desired event>      Index of the event you desire to broadcast in the
                                  array of events. When the server creates the
                                  EquipmentResourceImplementation, it passes an array
                                  of Events as a parameter to the constructor. This
                                  array is created in the method named createEvents.
                                  Specify the index of the event that you want
                                  broadcast periodically by the server.

    <interval between broadcasts> The time period between successive broadcasts in
                                  milliseconds.
```

For example:

```
java Server.SingleResourceDemoServer 0 3000
```

In this case, event 0 (relative to the array created in createEvents) is chosen to be broadcast every 3 seconds.

### 4.4.4 Starting the Client

The syntax for running the client is as follows:

```
java Client.EquipmentObserver <host of server>
    <host of server>      The name of the host where the server resides. You may use
                          localhost if the server is on the same machine.
```

For example:

```
java Client.EquipmentObserver localhost
```

Following is a transcript of client operation (characters in boldface indicate user input):

```
C:\users\default>java Client.EquipmentObserver localhost
Binding to :EquipSrv on localhost
[New IIOp Connection (localhost,IT_daemon, null,null,pid=0) ]
[New IIOp Connection (localhost,EquipSrv, null,null,pid=0) ]

Equipment Resource Description = New Equipment
Changing description to 'New Contraption'
Equipment Resource Description = New Contraption
Equipment Resource Function = Make Something
Equipment Resource Serial Number = A7B85JWX15324
The Equipment Resource is AVAILABLE.
```

EquipmentResource New Contraption broadcasts the following events:  
 Event 0: NEWEQUIPNEEDSMAINT New Equipment needs maintenance.  
 Event 1: NEWEQUIPISAVAIL New Equipment is available.  
 Event 2: NEWEQUIPISUNAVAIL New Equipment is unavailable.  
 Event 3: NEWEQUIPWAFERENTERED A wafer entered New Equipment.  
 Event 4: NEWEQUIPWAFEREXITED A wafer exited New Equipment.

Which event do you want to subscribe to? 0

Successfully subscribed to NEWEQUIPNEEDSMAINT

Processing events...

[ 1976: New Connection (GDAVIS:1979) ]

Event received (NEWEQUIPNEEDSMAINT): New Equipment needs maintenance.

Event received (NEWEQUIPNEEDSMAINT): New Equipment needs maintenance.

Event received (NEWEQUIPNEEDSMAINT): New Equipment needs maintenance.

Event received (NEWEQUIPNEEDSMAINT): New Equipment needs maintenance.

[ End of IOP connection (localhost:2000) ]

Here is a transcript of server operation for the same client session:

C:\users\default>**java Server.SingleResourceDemoServer 0 3000**

EquipSrv: impl is ready.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

[New Connection (localhost,IT\_daemon, \*,gdavis,pid=-889223576) ]

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

[ EquipSrv: New Connection (localhost:1978) ]

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

About to broadcast event: NEWEQUIPNEEDSMAINT to all subscribers.

Received subscription request from Notification.\_EventSubscriberStub@1ef9c6 to event NEWEQUIPNEEDSMAINT

Notification.\_EventSubscriberStub@1ef9c6 added as subscriber to event NEWEQUIPNE

**EDSMaint**

About to broadcast event: NEW EQUIPMENT NEEDS MAINT to all subscribers.

About to push an event notification across the wire.

[New IOP Connection (gdavis,1976, null,null,pid=0) ]

About to broadcast event: NEW EQUIPMENT NEEDS MAINT to all subscribers.

About to push an event notification across the wire.

About to broadcast event: NEW EQUIPMENT NEEDS MAINT to all subscribers.

About to push an event notification across the wire.

About to broadcast event: NEW EQUIPMENT NEEDS MAINT to all subscribers.

About to push an event notification across the wire.

## 4.5 Conclusions

After having successfully developed an implementation of the IDL subset derived from the Milestone 1 IDL, the team believes that the IDL is reasonable and that implementation of factory systems using CORBA-compliant OBEM components is feasible. The fact that the team successfully developed a sample implementation of one of the more challenging and risky portions of the IDL, together with previous experience in developing implementations of CORBA components, also contributed to this conclusion.

## 4.6 For More Information

A good introduction to implementing CORBA servers in Java is *Client/Server Programming with JAVA and CORBA*, by Robert Orfali and Dan Harkey (New York: John Wiley & Sons, Inc., 1997). It is not a complete treatment, but is a good starting point for learning more. For more information, refer to the documentation distributed with a commercial CORBA product.

# 5 MAPPING OBEM TO DCOM TECHNOLOGY

## 5.1 Introduction

DCOM is the other ORB that must be considered. CORBA is supported on more platforms and with more languages (see below), and while it is thought by many to be technically superior to DCOM<sup>8</sup>, the latter cannot be ignored, because it is a Microsoft standard.

The DCOM view of objects is similar to CORBA's, mainly in that both separate interface from implementation. Their interfaces are defined by IDL (Microsoft IDL for DCOM and OMG IDL for CORBA). Objects that support the interfaces can be implemented in several different languages. They differ often in terminology and paradigm, and sometimes use the same terminology with different meanings.

CORBA ORBs are available for several UNIX platforms, as well as Windows NT and Windows 95, whereas the DCOM operating system support has been mostly limited to Windows NT and Windows 95 platforms. (Microsoft has announced it would support several

---

<sup>8</sup> See, for example, Lewandowski, Scott M., "Frameworks for Component-Based Client/Server Computing," *ACM Computing Surveys*, vol. XXX, no. 1 (New York: ACM, 1998) 3-27, and Orfali and Harkey, *Client/Server Programming with JAVA and CORBA*, 331-337. But, for a different opinion see Grimes, Richard, *Professional DCOM Programming* (Birmingham, England: Wrox Press, 1997) 47.

UNIX platforms, however.) DCOM primarily supports development in C++ and Java, while CORBA language bindings are defined for C, C++, Java, Ada, Cobol, and Smalltalk. (The authors are unaware of actual products supporting C++, Java, and Smalltalk).

## 5.2 Mapping Considerations

Component Object Model (COM) is the object model used by DCOM. How some key elements of the OBEM object model could be mapped to the COM object model is discussed below.

It should be noted that there is a special flavor of COM interface, called “Automation,” that traditionally has had some important differences from standard COM interfaces. It is not believed that Automation interfaces are a particularly good match for OBEM, primarily because they traditionally have limited the data types that could be passed to methods and returned by them. Although this may no longer be the case (COM terminology is difficult to pin down because it changes over time), this discussion of COM technology is limited to non-Automation interfaces; some of the statements made in this section, while true of standard COM interface technology may not be true of Automation interface technology.

### 5.2.1 Objects

OBEM objects map reasonably well to Microsoft IDL interfaces. Some areas of possible concern include object navigation and inheritance.

#### 5.2.1.1 Object Navigation

Due to the fact that OBEM portrays physical equipment using objects that have relationships with other objects<sup>9</sup>, it is necessary for an implementation technology to support the definition of methods that return handles to other objects<sup>10</sup>. This is workable in COM. Sometimes, too, one might wish to have a collection of such objects returned. This can be done with an array or through a collection object or iterator object<sup>11</sup>.

#### 5.2.1.2 Inheritance

In Microsoft IDL, inheritance is handled differently from OMG IDL. Only single inheritance is allowed in Microsoft IDL. At first, this seems to imply that one could not, for example, add RecipeExecutor capability to EquipmentModule without adding the appropriate methods definitions directly to the EquipmentModule interface definition. It turns out that COM can handle this in a more elegant way: a COM implementation object actually can implement more than one interface not in the same inheritance hierarchy. Through its IUnknown interface, an interface that all COM objects implement, one can ask an object for its other interfaces.

### 5.2.2 Attributes

OBEM attributes can be mapped to *get* and *set* methods in Microsoft IDL.

---

<sup>9</sup> Sometimes these are relationships between superior or aggregate objects and subordinate or constituent objects, such as the relationship of a piece of equipment to its equipment modules; sometimes these are relationships between peer objects, such as the relationship between carrier ports and port locations.

<sup>10</sup> Of course, it is reasonable to consider an interface wherein an object returns a description of its constituent objects rather than handles to them. Such an interface might require some type of path descriptor (similar to a file path) to specify an arbitrarily nested aggregation hierarchy.

<sup>11</sup> An iterator object traverses a collection of objects and hands back a different object on each successive call to `next_object`. Of course, it must have some type of method to let the client know when there are none left.

### 5.2.3 Services

OBEM services generally will map nicely to Microsoft IDL. Microsoft IDL methods return only error/status information through the method return value. Any data returned is returned through an out parameter. In the case of optional services, the standard COM practice would be to return a particular error code through the return value, indicating that the method is not implemented.

Although exceptions are not specified in OBEM, the project's OMG IDL defines several and specifies the operations that can raise them. Although exceptions can be thrown and caught across interfaces, under certain conditions, in the COM model, one cannot declare in the IDL that a particular method can raise a particular exception.

Still remaining will be some of the problems found in mapping OBEM to CORBA, namely optional attributes, optional services, and optional parameters. To some extent, these OBEM features (especially optional services) can be handled by offering alternative interfaces. This is a more common practice in DCOM than in CORBA. When this is not acceptable, these would have to be handled on an ad hoc basis as they are in the CORBA implementation. Optional parameters also can be handled by specifying overloaded methods (methods that have the same name but different parameter signatures).

### 5.2.4 Data Types

OBEM data types will map to DCOM data types in a manner very similar to the OBEM-to-CORBA data type mapping. Numeric attributes and parameters may be mapped to a user-defined union, as the team did in its OMG IDL. They also may be mapped to a standard COM union called VARIANT.

### 5.2.5 Alarm and Event Reporting and Notification

OBEM specifies SEMI E53 Event Reporting Standard as the method of reporting events. However, in the team's OMG IDL, a new event reporting mechanism was introduced that is more in line with CORBA. DCOM has its own event notification mechanism, called Connectable Objects technology. This technology provides a general method for a COM (server) object to talk back to its client.

In this technology, the server object is called a connectable object. A connectable object has an interface named IConnectionPointContainer. A client can use IConnectionPointContainer to find out if the object knows how to send notifications to its desired interface. For example, a client wants to subscribe for notifications and be notified using the user-defined IEventSubscriber interface. The IConnectionPointContainer interface could be used to ask the object if it has a *connection point* object that knows how to hand a notification to an IEventSubscriber interface. If the server object has a connection point that is familiar with IEventSubscriber, the query will return a pointer to the IConnectionPoint interface for the proper connection point.

The client then would call the Advise method on the IConnectionPoint interface to pass a pointer to its IUnknown interface. Since the connection point handles only the IEventSubscriber interface (each connection point knows how to hand a notification to exactly one interface), it would ask the IUnknown interface for a pointer to the IEventSubscriber interface. It then would use this interface pointer to notify the client of an event. A similar mechanism could be used for alarms.

### 5.3 Level of Effort

Starting from scratch, a knowledgeable team of COM/DCOM developers could be expected to take approximately three weeks (as it did the project team) to develop Microsoft IDL for OBEM. This would include time to digest the OBEM standard and other related material. Of course, the project team had access to an OBEM task force member, and one team member had previous knowledge of the role, purpose, and intent of OBEM.

However, it seems reasonable to assume that a new team of developers would use the OMG IDL developed by the project team. Since OMG IDL is similar to C++ and many COM/DCOM developers are familiar with C++, it seems that the project's OMG IDL would provide a head start, shaving at least a week off IDL development time.

### 5.4 Conclusions

This has been a brief, high level review of how OBEM could be mapped to COM. Although actual mapping has not been done, research indicates that it is feasible.

### 5.5 For More Information

To learn about COM/DCOM at a relatively high level, recommended reading includes:

- David Chappell, *Understanding ActiveX and OLE*, Redmond, WA: Microsoft Press, 1996
- Orfali and Harkey, *Client/Server Programming with JAVA and CORBA*. This book compares CORBA and COM/DCOM.
- Lewandowski, Scott M., "Frameworks for Component-Based Client/Server Computing," *ACM Computing Surveys*, no. 1 (New York: ACM, 1998). This article also compares CORBA and COM/DCOM.
- Grimes, Richard, *Professional DCOM Programming*, Birmingham, England: Wrox Press, 1997. This volume gives good detail on COM/DCOM client/server implementation while tying it into the wider application technology picture.



```

};
exception ResourceCannotBeMadeAvailable {
    EquipmentResource resource;
    EquipmentResourceState state;
    string faultCondition;
};
exception ResourceCannotBeMadeUnavailable {
    EquipmentResource resource;
    EquipmentResourceState state;
    string faultCondition;
};
exception ResourceNotFound {
    EquipmentResource resource;
};

// -----
// OBEM Attributes
// -----

// Description
string getDescription();
void setDescription(in string description)
    raises (OBEM::StringParameterTooLong);

// Function
string getFunction();

// Serial Number
string getSerialNumber();

// Available
boolean isAvailable();

// -----
// OBEM Services
// -----

// Make Unavailable/Make Available
boolean supportsChangingAvailability();
void makeUnavailable()
    raises (ResourceCannotBeMadeUnavailable,
OBEM::OptionalOperationNotSupported);
void makeAvailable()
    raises (ResourceCannotBeMadeAvailable,
OBEM::OptionalOperationNotSupported);

// Add Sensor Actuator Device/Remove Sensor Actuator Device
boolean supportsAddSensorActuatorDevice();
void addSensorActuatorDevice(
    in SensorActuatorController::SensorActuatorDevice device)
    raises (DeviceCannotBeAdded,
OBEM::OptionalOperationNotSupported);
void removeSensorActuatorDevice(
    in SensorActuatorController::SensorActuatorDevice device)
    raises (DeviceNotFound, OBEM::OptionalOperationNotSupported);

// Add Resource/Remove Resource

```

```

        boolean supportsAddResource();
        void addResource(in EquipmentResource resource)
            raises (ResourceCannotBeAdded,
OBEM::OptionalOperationNotSupported);
        void removeResource(in EquipmentResource resource)
            raises (ResourceNotFound,
OBEM::OptionalOperationNotSupported);

        // -----
        // Access To Owned Objects
        // -----
        EquipmentResourceSequence getOwnedResources();
        SensorActuatorController::SensorActuatorDeviceSequence
            getSensorActuatorDevices();
        Transport::MaterialLocationSequence getMaterialLocations();
    };

//
=====
//
//                               INTERFACE Equipment Module
//
=====
interface EquipmentModule:
    EquipmentResource,
    ARAMS::ARAMSObject,
    RMS::RecipeExecutor {

    // -----
    // Types
    // -----
    enum OperationsState {
        os_Aborted, os_Aborting, os_Active, os_ActiveService,
        os_Idle, os_IdleWithAlarms, os_Inactive, os_Initialization,
        os_Operational, os_Pause, os_Paused, os_Pausing, os_PowerDown,
        os_Stopped, os_Stopping
    };
    enum ProcessType {
        pt_Process, pt_Measurement, pt_Transport, pt_Storage
    };

    // -----
    // Exceptions
    // -----
    exception ProcessCapabilityNotFound {
        string processCapability;
    };
    exception ModuleCannotBeStarted {};
    exception ModuleCannotBeStopped {};
    exception ModuleCannotBeAborted {};
    exception ModuleCannotBePaused {};
    exception ModuleCannotBeResumed {};

    // -----
    // OBEM Attributes
    // -----

```

```

// Module Name
string getModuleName();
void setModuleName(in string moduleName)
    raises (OBEM::StringParameterTooLong);

// Manufacturer
string getManufacturer();

// Model
string getModel();

// Model Revision
string getModelRevision();

// Software Version
OBEM::StringSequence getSoftwareVersion();

// Holding Capacity
unsigned long getHoldingCapacity();

// Units
string getUnits();

// Process Type
ProcessType getProcessType();
void setProcessType(in ProcessType type);

// Process Capability
OBEM::StringSequence getProcessCapabilities();
void addProcessCapability(in string processCapability)
    raises (OBEM::StringParameterTooLong);
void removeProcessCapability(in string processCapability)
    raises (ProcessCapabilityNotFound);

// Active Recipes
boolean supportsRecipeExecution();
OBEM::StringSequence getActiveRecipes()
    raises (OBEM::OptionalOperationNotSupported);

// Operations State
OperationsState getOperationsState();

// Previous Operations State
OperationsState getPreviousOperationsState();

// -----
// OBEM Services
// -----

// Start
void start() raises (ModuleCannotBeStarted);

// Stop
void stop() raises (ModuleCannotBeStopped);

// Abort

```

```

void abort() raises (ModuleCannotBeAborted);

// Pause
void pause() raises (ModuleCannotBePaused);

// Resume
void resume() raises (ModuleCannotBeResumed);

// -----
// Access to owned Objects
// -----
boolean supportsClock();
Time::Clock getClock() raises (OBEM::OptionalOperationNotSupported);
};

//
=====
//
//                               INTERFACE Equipment
//
=====
interface Equipment: EquipmentModule {

// -----
// Exceptions
// -----
exception EquipmentCannotBeCleanedUp {
    EquipmentModule::OperationsState currentState;
    string reason;
};

// -----
// OBEM Attributes
// -----

// Assigned-Operators
OBEM::StringSequence getAssignedOperators();

// Equipment Name
string getEquipmentName();
void setEquipmentName(in string equipmentName)
    raises (OBEM::StringParameterTooLong);

// Factory-Location
string getFactoryLocation();
void setFactoryLocation(in string factoryLocation)
    raises (OBEM::StringParameterTooLong);

// -----
// OBEM Services
// -----

// Cleanup
void cleanup() raises (EquipmentCannotBeCleanedUp);

// -----

```

```

        // Access To Owned Objects
        // -----
        Transport::CarrierPortSequence getCarrierPorts();
    };
};
#endif

```

### A.1.2 Notification.idl

```

#ifndef Notification_idl
#define Notification_idl
#include <CosEventComm.idl>

module Notification {

//
=====
//
//
//
=====
    enum AlarmState {as_On, as_Off, as_Warning};
    //
    // This is the type that an AlarmGenerator
    // should use to notify an AlarmSubscriber.
    //
    struct AlarmNotification {
        string alarmID;
        string<80> alarmText;
        AlarmState alarmState;
    };
    interface AlarmSubscriber: CosEventComm::PushConsumer {
    };

//
=====
//
//
//
=====

    // -----
    // Types
    // -----
    struct Alarm {
        string alarmID;
        string<80> alarmText;
        AlarmState alarmState;
        boolean alarmReportingIsEnabled;
    };
    typedef sequence<Alarm> AlarmSequence;

    // -----
    // The Interface

```

```

// -----
interface AlarmGenerator {

    // -----
    // Exceptions
    // -----
    exception AlarmDoesNotExist {string alarmID;};
    exception AlarmTextTooLong {
        string alarmID;
        string alarmText;
        unsigned short maximumLength;
    };

    // -----
    // OBEM Attributes
    // -----

    // Alarms
    AlarmSequence getAlarms();
    Alarm getAlarm(in string alarmID) raises (AlarmDoesNotExist);

    // Alarm Text
    void setAlarmText(in string alarmID, in string text)
        raises (AlarmTextTooLong, AlarmDoesNotExist);

    // Reporting Enablement
    boolean alarmReportingIsEnabled(in string alarmID)
        raises (AlarmDoesNotExist);
    void enableAlarmReporting(in string alarmID) raises (AlarmDoesNotExist);
    void disableAlarmReporting(in string alarmID) raises (AlarmDoesNotExist);
    void enableReportingOfAllAlarms();
    void disableReportingOfAllAlarms();

    // -----
    // Subscription
    // -----
    void subscribeToAlarm(in string alarmID, in AlarmSubscriber subscriber)
        raises (AlarmDoesNotExist);
    void unsubscribeFromAlarm(in string alarmID, in AlarmSubscriber
subscriber)
        raises (AlarmDoesNotExist);
    };

//
=====
//                                     INTERFACE Event Subscriber
//
=====
//
// This is the type that an EventGenerator
// should use to notify an EventSubscriber.
//
struct EventNotification {
    string eventID;
    string<80> eventText;
};

```

```

interface EventSubscriber: CosEventComm::PushConsumer {
};

//
=====
=====
//                                     INTERFACE Event Generator
//
=====
=====

// -----
// Types
// -----
struct Event {
    string eventID;
    string<80> eventText;
    boolean eventReportingIsEnabled;
};
typedef sequence<Event> EventSequence;

// -----
// The Interface
// -----
interface EventGenerator {

    // -----
    // Exceptions
    // -----
    exception EventDoesNotExist {string eventID;};
    exception EventTextTooLong {
        string eventID;
        string eventText;
        unsigned short maximumLength;
    };

    // -----
    // OBEM Attributes
    // -----

    // Events
    EventSequence getEvents();
    Event getEvent(in string eventID) raises (EventDoesNotExist);

    // Event Text
    void setEventText(in string eventID, in string text)
        raises (EventTextTooLong, EventDoesNotExist);

    // Reporting Enablement
    boolean eventReportingIsEnabled(in string eventID)
        raises (EventDoesNotExist);
    void enableEventReporting(in string eventID) raises (EventDoesNotExist);
    void disableEventReporting(in string eventID) raises (EventDoesNotExist);
    void enableReportingOfAllEvents();
    void disableReportingOfAllEvents();

    // -----
    // Subscription

```

```

// -----
void subscribeToEvent(in string eventID, in EventSubscriber subscriber)
    raises (EventDoesNotExist);
void unsubscribeFromEvent(in string eventID, in EventSubscriber
subscriber)
    raises (EventDoesNotExist);
};
};
#endif

```

### A.1.3 OBEM.idl

```

#ifndef OBEM_idl
#define OBEM_idl

module OBEM {

// -----
// Types
// -----
typedef sequence<string> StringSequence;
typedef sequence<any> AnySequence;
typedef sequence<unsigned short> UShortSequence;
typedef sequence<octet> OctetSequence;
typedef double Timestamp;
enum NumericType {nt_Integer, nt_FloatingPoint};
union Numeric switch (NumericType) {
    case nt_Integer: long intValue;
    case nt_FloatingPoint: double floatValue;
};
enum NumericOrTextType {nott_Numeric, nott_Text};
union NumericOrText switch (NumericOrTextType) {
    case nott_Numeric: Numeric numericValue;
    case nott_Text: string stringValue;
};

// -----
// Exceptions
// -----
exception StringParameterTooLong {
    string parameterValue;
    unsigned long maximumLength;};
exception OptionalOperationNotSupported {};
exception InvalidNumericValue {any value;};
exception NumericValueOutOfRange {any value;};
exception IncorrectNumericType {
    any value;
    string correctType;
};
exception TimestampUnreasonable {Timestamp value;};
exception FormattedTimestampUnreasonable {string value;};
exception InvalidTimestampFormat {string format;};
exception TimestampFormattedImproperly {string timestamp;};
};

```

```

// -----
// Stub modules
// -----
#ifndef ARAMS_idl
module ARAMS {
    interface ARAMSObject {
    };
};
#endif

#ifndef RMS_idl
module RMS {
    interface RecipeExecutor {
    };
};
#endif

#endif

```

#### A.1.4 SAC.idl

```

#ifndef SensorActuatorController_idl
#ifndef SensorActuatorController_idl
#define SensorActuatorController_idl

#include "OBEM.idl"

#include <CosEventComm.idl>

module SensorActuatorController {

    // -----
    // Common Exceptions
    // -----
    exception InvalidUnits {string units;};

    // -----
    // Common Types
    // -----
    //
    // This is the type that a SensorReportGenerator
    // should use to report sensor values to a SensorReportSubscriber.
    //
    struct SensorReport {
        any timestampedValue;
    };

    //
    =====
    //
    //
    //
    =====
    interface SensorReportSubscriber: CosEventComm::PushConsumer {

```

```

};

//
=====
//
//                               INTERFACE Sensor Report Generator
//
=====
interface SensorReportGenerator {

    // -----
    // OBEM Attributes
    // -----

    // Report Enabling
    boolean supportsReportEnableDisable();
    boolean isReportingEnabled() raises
(OBEM::OptionalOperationNotSupported);
    void enableReporting() raises (OBEM::OptionalOperationNotSupported);
    void disableReporting() raises(OBEM::OptionalOperationNotSupported);

    // Report Rate
    double getReportRate();

    // -----
    // Subscription
    // -----
    void subscribeToReports(in SensorReportSubscriber subscriber);
    void unsubscribeFromReports(in SensorReportSubscriber subscriber);
};

//
=====
//
//                               INTERFACE Sensor
//
=====
interface Sensor: SensorReportGenerator {

    // -----
    // OBEM Attributes
    // -----

    // Description
    string<20> getDescription();
    void setDescription(in string<20> newDescription);

    // Read Rate
    double getReadRate();

    // Timestamped Value
    any getValue();
};
typedef sequence<Sensor> SensorSequence;

```

```

//
=====
=====
//                                     INTERFACE Discrete Sensor
//
=====
=====
    struct TimestampedDiscreteValue {
        OBEM::Timestamp timestamp;
        string value;
    };
    interface DiscreteSensor: Sensor {

        // -----
        // OBEM Attributes
        // -----

        // Range
        OBEM::StringSequence getRange();

        // Timestamped Value
        TimestampedDiscreteValue getDiscreteValue();
    };

//
=====
=====
//                                     INTERFACE Scalar Sensor
//
=====
=====
    struct TimestampedScalarValue {
        OBEM::Timestamp timestamp;
        OBEM::Numeric value;
    };
    interface ScalarSensor: Sensor {

        // -----
        // OBEM Attributes
        // -----

        // Units/Possible Units
        string getUnits();
        boolean supportsChangingUnits();
        void setUnits(in string units) raises (InvalidUnits,
OBEM::OptionalOperationNotSupported);
        OBEM::StringSequence getPossibleUnits() raises
(OBEM::OptionalOperationNotSupported);

        // Interval
        string getInterval();

        // Timestamped Value
        TimestampedScalarValue getScalarValue();

```

```

        // Nominal Value/Delta Value
        boolean supportsValueChangeReporting();
        OBEM::Numeric getNominalValue() raises
(OBEM::OptionalOperationNotSupported);
        void setNominalValue(in OBEM::Numeric value)
            raises (
                OBEM::InvalidNumericValue,
                OBEM::NumericValueOutOfRange,
                OBEM::IncorrectNumericType,
                OBEM::OptionalOperationNotSupported
            );

        OBEM::Numeric getDeltaValue() raises
(OBEM::OptionalOperationNotSupported);
        void setDeltaValue(in OBEM::Numeric value)
            raises (
                OBEM::InvalidNumericValue,
                OBEM::NumericValueOutOfRange,
                OBEM::IncorrectNumericType,
                OBEM::OptionalOperationNotSupported
            );
    };

//
=====
=====
//
//                                     INTERFACE Vector Sensor
//
=====
=====
    struct TimestampedVectorValue {
        OBEM::Timestamp timestamp;
        OBEM::AnySequence value;
    };
    interface VectorSensor: Sensor {

        // -----
        // OBEM Attributes
        // -----

        // Timestamped Value
        TimestampedVectorValue getVectorValue();

        // Data Dimensions
        OBEM::UShortSequence getDataDimensions();
    };

//
=====
=====
//
//                                     INTERFACE Actuator
//
=====
=====
    interface Actuator {

```

```

// -----
// OBEM Attributes
// -----

// Description
string<20> getDescription();
void setDescription(in string<20> newDescription);

// Timestamped Value
any getValue(); // returns a XTimestampedValue
};
typedef sequence<Actuator> ActuatorSequence;

//
=====
//
//                               INTERFACE Discrete Actuator
//
=====
=====
interface DiscreteActuator: Actuator {

// -----
// OBEM Attributes
// -----

// Range
OBEM::StringSequence getRange();

// Timestamped Value
TimestampedDiscreteValue getDiscreteValue();

};

//
=====
=====
//
//                               INTERFACE Scalar Actuator
//
=====
=====
interface ScalarActuator: Actuator {

// -----
// OBEM Attributes
// -----

// Interval
string getInterval();

// Units/Possible Units
string getUnits();
boolean supportsChangingUnits();
OBEM::StringSequence getPossibleUnits() raises
(OBEM::OptionalOperationNotSupported);
void setUnits(in string units) raises (InvalidUnits,
OBEM::OptionalOperationNotSupported);

```

```

        // Timestamped Value
        TimestampedScalarValue getScalarValue());
};

//
=====
//
//                               INTERFACE Vector Actuator
//
=====
interface VectorActuator: Actuator {

    // OBEM Attributes

    // Timestamped Value
    TimestampedVectorValue getVectorValue();

    // Data Dimensions
    OBEM::UShortSequence getDataDimensions();
};

//
=====
//
//                               INTERFACE Vector Actuator
//
=====

// -----
// Types
// -----
struct Dataltem {
    string name;
    OBEM::NumericOrText value;
    string units;
    OBEM::StringSequence possibleUnits;
    OBEM::StringSequence range;
};
typedef sequence<Dataltem> DataltemSequence;

struct ProgramDirectoryEntry {
    string programID;
    OBEM::Timestamp timestamp;
};
typedef sequence<ProgramDirectoryEntry> ProgramDirectory;

// -----
// The Interface
// -----
interface SensorActuatorDevice {

    // -----
    // Types
    // -----
    struct ProgramDirectoryEntry {

```

```

        string programID;
        string description;
        OBEM::Timestamp time;
};
typedef sequence<ProgramDirectoryEntry> ProgramDirectory;
enum DeviceStatus {
    ds_INIT_OR_SELF_TEST, ds_IDLE, ds_SELF_TEST_EXC,
    ds_EXECUTING, ds_ABORT_FROM_IDLE_OR_EXE,
    ds_ABORT_FROM_INIT_OR_SELF_TEST_OR_SELF_TEST_EXC
};

// -----
// Exceptions
// -----
exception DataItemNotFound {string name;};
exception InvalidAlgorithmID {string id;};
exception InvalidNumberOfItemsValue {unsigned short value;};
exception InvalidStartValue {unsigned short value;};
exception ProgramTooLong {unsigned long length;};

// -----
// OBEM Attributes
// -----

// Algorithm ID
string getAlgorithmID();
boolean supportsChangingAlgorithmID();
void setAlgorithmID(in string id)
    raises (
        InvalidAlgorithmID,
        OBEM::StringParameterTooLong,
        OBEM::OptionalOperationNotSupported);

// Number of Data Items
unsigned long getNumberOfDataItems();

// Data Items
DataItemSequence getDataItemRange(
    in unsigned short startIndex,
    in unsigned short numberOfItems
)
    raises (InvalidStartValue, InvalidNumberOfItemsValue);
DataItemSequence getDataItems();
boolean supportsChangingUnits();
void setDataItemUnits(in string name, in string units)
    raises (
        InvalidUnits,
        DataItemNotFound,
        OBEM::OptionalOperationNotSupported
    );

// Device Type
string<8> getDeviceType();

// Device Manufacturer ID
string<20> getDeviceMfgID();

```

```

// Device Status
DeviceStatus getDeviceStatus();

// Hardware Revision
string<5> getHardwareRevision();

// Model Number
string<20> getModelNumber();

// Sensor IDs
OBEM::StringSequence getSensorIDs();

// Software Revision
string<5> getSoftwareRevision();

// Timestamp
OBEM::Timestamp getTimestamp();

// -----
// Access to Owned Objects
// -----

// External Actuators
boolean supportsExternalActuators();
ActuatorSequence getExternalActuators() raises
(OBEM::OptionalOperationNotSupported);

// External Sensors
boolean supportsExternalSensors();
SensorSequence getExternalSensors() raises
(OBEM::OptionalOperationNotSupported);

// -----
// OBEM Services
// -----

// Download Sensor Program
void downloadSensorProgram(
    in string programID,
    in string description,
    in OBEM::OctetSequence program,
    in OBEM::Timestamp time
) raises (
    InvalidAlgorithmID,
    ProgramTooLong,
    OBEM::TimestampUnreasonable,
    OBEM::StringParameterTooLong
);

// Select Sensor Program
boolean supportsProgramLibrary();
void selectSensorProgram(in string programID);

// Get Sensor Program Directory
unsigned short getNumberOfSensorPrograms();
ProgramDirectory getSensorProgramDirectory();
};

```





```

        // Location ID
        string getLocationID();

        // Current State
        MaterialLocationState getCurrentState();

        // Material ID
        string getMaterialID();
    };
    typedef sequence<MaterialLocation> MaterialLocationSequence;

//
=====
//
//                                     INTERFACE Carrier Location
//
=====
=====
    interface CarrierLocation: MaterialLocation {

        // -----
        // OBEM Attributes
        // -----

        // Carrier ID
        string getCarrierID();

        // Is Substrate Port?
        boolean isSubstratePort();
    };

//
=====
//
//                                     INTERFACE Port Location
//
=====
=====
    interface CarrierPort;
    interface PortLocation: CarrierLocation {

        // -----
        // Access to Related Objects
        // -----

        // Assigned Port
        CarrierPort getAssignedPort();
    };
    typedef sequence<PortLocation> PortLocationSequence;

//
=====
//
//                                     INTERFACE Carrier Port
//
=====
=====

```

```

interface CarrierPort {

    // -----
    // Types
    // -----
    enum PortAvailabilitySubstate {
        pas_AllocatedToEquipment, pas_AllocatedToTransfer,
        pas_AssignedToLoad, pas_AssignedToUnload, pas_LoadActive,
        pas_NotAllocated, pas_NotAssigned, pas_PortAvailable,
        pas_PortUnavailable, pas_UnloadActive
    };
    enum PortReservationSubstate {
        prs_PortNotReserved, prs_PortReserved,
        prs_ReservedForLoad, prs_ReservedForUnload
    };
    enum PortUsage {
        pu_Input, pu_Output, pu_InputOutput
    };

    // -----
    // OBEM Attributes
    // -----

    // Port Alarm ID
    OBEM::Numeric getPortAlarmID();

    // Port Alarm Text
    string getPortAlarmText();

    // Port Availability
    PortAvailabilitySubstate getPortAvailability();

    // Previous Port Availability
    PortAvailabilitySubstate getPreviousPortAvailability();

    // Port Reservation Substate
    boolean supportsPortReservation();
    PortReservationSubstate getPortReservationSubstate()
        raises (OBEM::OptionalOperationNotSupported);

    // Port Usage
    PortUsage getPortUsage();
    boolean supportsChangingPortUsage();
    void setPortUsage(in PortUsage portUsage)
        raises (OBEM::OptionalOperationNotSupported);

    // -----
    // Access to Related Objects
    // -----

    // Assigned Location
    PortLocation getAssignedLocation();

    // Available Locations
    PortLocationSequence getAvailableLocations();

};

```

```

        typedef sequence<CarrierPort> CarrierPortSequence;
    };
#endif

```

## A.2 Subset of IDL Used for Java Implementation

This section contains the IDL used in the demo implementation.

### A.2.1 Equip.idl

```

#ifndef Equip_idl
#define Equip_idl

#include "Notification.idl"

module Equip {

    // -----
    // Exceptions
    // -----

    // String Parameter Too Long
    exception StringParameterTooLong {
        string parameterValue;
        unsigned long maximumLength;
    };

    //
    =====
    //
    //
    //
    =====
    interface EquipmentResource: Notification::EventGenerator {

        // -----
        // OBEM Attributes
        // -----

        // Description
        string getDescription();
        void setDescription(in string description)
            raises (StringParameterTooLong);

        // Function
        string getFunction();

        // Serial Number
        string getSerialNumber();

        // Available
        boolean isAvailable();

    };
}

```

```
};
#endif
```

### A.2.2 Notification.idl

```
#ifndef Notification_idl
#define Notification_idl
#include <CosEventComm.idl>

module Notification {

//
=====
//
//                               INTERFACE Event Subscriber
//
=====
//
// This is the type that an EventGenerator
// should use to notify an EventSubscriber.
//
struct EventNotification {
    string eventID;
    string<80> eventText;
};
interface EventSubscriber: CosEventComm::PushConsumer {
};

//
=====
//
//                               INTERFACE Event Generator
//
=====
// -----
// Types
// -----
struct Event {
    string eventID;
    string<80> eventText;
    boolean eventReportingIsEnabled;
};
typedef sequence<Event> EventSequence;

// -----
// The Interface
// -----
interface EventGenerator {

// -----
// Exceptions
// -----
exception EventDoesNotExist {string eventID;};
```

```

// -----
// OBEM Attributes
// -----

// Events
EventSequence getEvents();
Event getEvent(in string eventID) raises (EventDoesNotExist);

// -----
// Subscription
// -----
void subscribeToEvent(in string eventID, in EventSubscriber subscriber)
    raises (EventDoesNotExist);
void unsubscribeFromEvent(in string eventID, in EventSubscriber
subscriber)
    raises (EventDoesNotExist);
};
};
#endif
```

## APPENDIX B JAVA Source Code

This section contains the Java source code used in the demo implementation.

### B.1 **EquipmentResourceImplementation.java**

```

package Equip;

import org.omg.CORBA.Any;
import org.omg.CORBA.ORB;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;
import CosEventComm.Disconnected;
import Notification.Event;
import Notification.EventNotification;
import Notification.EventNotificationHelper;
import Notification.EventSubscriber;
import Notification.EventGeneratorPackage.EventDoesNotExist;

/**
 * Implements the EquipmentResource interface for the SingleResourceDemoServer.
 * This is a simplified version of the EquipmentResource specified in the full
 * IDL delivered with the OBEM Proof of Concept Project (under Agreement #78050122).
 *
 * @author Gordon Davis
 * @version 0.0
 */
public class EquipmentResourceImplementation extends _EquipmentResourceImplBase {

    // OBEM API
    /**
     * Returns the description of this resource.
     *
     * @return      the description of this resource.
     * @see         String
     */
    public String getDescription() {
        return description;
    }

    /**
     * Sets the description of this resource.
     *
     * @param description      the new description
     * @exception Equip.StringParameterTooLong indicates that the given description is too long.
     */
    public void setDescription(String description) throws StringParameterTooLong {
        if (description.length() > 100)
            throw new StringParameterTooLong(description, 100);
        this.description = description;
        return;
    }
}

```

```

    }

    /**
     * Returns the function of this resource.
     *
     * @return          the function of this resource.
     *
     */
    public String getFunction() {
        return function;
    }

    /**
     * Returns the serial number of this resource.
     *
     * @return          the serial number of this resource.
     *
     */
    public String getSerialNumber() {
        return serialNumber;
    }

    /**
     * Returns the availability of this resource.
     *
     * @return          <code>true</code> if this resource is available;
     *                  <code>false</code> otherwise.
     *
     */
    public boolean isAvailable() {
        return isAvailable;
    }

    // Event Generator API
    /**
     * Returns all of the events this resource can broadcast.
     *
     * @return          all events
     * @see            Notification.Event
     *
     */
    public Event[] getEvents() {
        return events;
    }

    /**
     * Returns a particular event given an eventID.
     *
     * @param eventID  the ID of the event to be retrieved
     * @return          the desired event
     * @exception Notification.EventGeneratorPackage.EventDoesNotExist
     *                  denotes that the desired event does
     not exist
     *
     */
    public Event getEvent(String eventID) throws EventDoesNotExist {
        for (int index = 0; index < events.length; ++index) {

```

```

        if (events[index].eventID.equals(eventID))
            return events[index];
    }
    throw new EventDoesNotExist(eventID);
}

/**
 * Subscribes to a particular event.
 *
 * @param eventID          the ID of the event of which notification is desired
 * @param subscriber      the subscriber that wants to subscribe to the event
 * @exception Notification.EventGeneratorPackage.EventDoesNotExist
 *                        denotes that the desired event does
not exist
 * @see                   Notification.EventSubscriber
 */
public void subscribeToEvent(String eventID, EventSubscriber subscriber)
    throws EventDoesNotExist {
    System.out.println("Received subscription request from " + subscriber + " to event " +
        eventID);
    Event event = getEvent(eventID);
    Vector subscriberList = (Vector) subscribers.get(event);
    if (subscriberList == null) {
        subscriberList = new Vector(3);
        subscribers.put(event, subscriberList);
    }
    if (indexOfRemoteObjectInVector(subscriber, subscriberList) == -1)
        subscriberList.addElement(subscriber);
    System.out.println(subscriber + " added as subscriber to event " + event.eventID);
    return;
}

/**
 * Unsubscribes a subscriber from an event's subscriber list.
 *
 * @param eventID          the ID of the event of which notification is desired
 * @param subscriber      the subscriber that wants to subscribe to the event
 * @exception Notification.EventGeneratorPackage.EventDoesNotExist
 *                        denotes that the desired event does
not exist
 */
public void unsubscribeFromEvent(String eventID, EventSubscriber subscriber)
    throws EventDoesNotExist {
    System.out.println("Received unsubscribe request from " + subscriber + " from event " +
        eventID);
    Event event = getEvent(eventID);
    Vector subscriberList = (Vector) subscribers.get(event);
    if (subscriberList != null) {
        int subscriberIndex =
            indexOfRemoteObjectInVector(subscriber, subscriberList);
        if (subscriberIndex != -1) {
            subscriberList.removeElementAt(subscriberIndex);
            if (subscriberList.isEmpty()) {
                subscribers.remove(event);
            }
        }
    }
}

```

```

        }
        System.out.println(subscriber + " no longer subscribes to event " + event.eventID);
        return;
    }

// Constructors
/**
 * Default constructor; does nothing special.
 */
public EquipmentResourceImplementation() {}

/**
 * Constructor for creating a populated instance.
 *
 * @param description      the description of this resource
 * @param function         the function of this resource
 * @param serialNumber     the serial number of this resource
 * @param isAvailable      <code>true</code> if this resource should start out
 *                          available; <code>false</code>
otherwise
 * @param events          the events that this resource may broadcast
 */
public EquipmentResourceImplementation(
    String description,
    String function,
    String serialNumber,
    boolean isAvailable,
    Event[] events)
{
    this();
    // Transfer data from parameters to instance variables
    this.description = description;
    this.function = function;
    this.serialNumber = serialNumber;
    this.isAvailable = isAvailable;
    this.events = events;
}

/**
 * Constructor for creating a populated instance that broadcasts one specific event
 * at a regular interval. This method is specially devised to demonstrate the
 * broadcasting of events and aid in testing clients that receive event
 * notifications.
 *
 * @param description      the description of this resource
 * @param function         the function of this resource
 * @param serialNumber     the serial number of this resource
 * @param isAvailable      <code>true</code> if this resource should start out
 *                          available; <code>false</code>
otherwise
 * @param events          the events that this resource may broadcast
 * @param eventID         the ID of the event that should be broadcast periodically.
 * @param broadcastInterval the number of milliseconds between broadcasts of the event

```

```

*/
public EquipmentResourceImplementation(
    String description,
    String function,
    String serialNumber,
    boolean isAvailable,
    Event[] events,
    String eventID,
    int broadcastInterval)
{
    this(description, function, serialNumber, isAvailable, events);
    Event event = null;
    try {event = getEvent(eventID);}
    catch (EventDoesNotExist e) {}
    if (event != null) {

        // Anonymous inner classes can only access local variables that are final.
        final Event eventToBroadcast = event;
        final int _broadcastInterval = broadcastInterval;

        // Define thread that broadcasts the event every so often.
        thread = new Thread(new Runnable() {
            public void run() {
                while (true) {
                    try {
                        System.out.println("About to broadcast event: " +
                            eventToBroadcast.eventID + " to all
subscribers.");

                        broadcastEvent(eventToBroadcast);
                        Thread.sleep(_broadcastInterval);
                    }
                    catch (Disconnected e) {
                        e.printStackTrace();
                        System.exit(1);
                    }
                    catch (InterruptedException e) {
                        e.printStackTrace();
                        System.exit(1);
                    }
                }
            }
        });

        // Start the thread
        thread.start();
    }

    // Event Broadcasting
    /**
     * Broadcasts an event.
     *
     * @param eventID          the ID of the event of which notification is desired
     * @exception CosEventComm.Disconnected
     */
    protected void broadcastEvent(Event event)

```

```

throws Disconnected {

// Get the subscriber list for the event.
Vector subscriberList = (Vector) subscribers.get(event);
if (subscriberList != null) {

    // For each subscriber in the subscriber list...
    Enumeration enum = subscriberList.elements();
    while(enum.hasMoreElements()) {

        // Create the notification
        EventNotification notification =
            new EventNotification(event.eventID, event.eventText);

        // Create an Any
        Any param = ORB.init().create_any();
        EventNotificationHelper.insert(param, notification);

        // Broadcast it to the subscriber.
        System.out.println("About to push an event notification across the wire.");
        ((EventSubscriber) enum.nextElement()).push(param);
    }
}
return;
}

// Utility methods
/**
 * Find a remote object in a <code>Vector</code>.
 *
 * @param remoteObject      the remote object (reference)
 * @param vector            the vector in which to search for the
 *                          remote object
 * @return                  the index of the remote object in the vector (-1
 *                          if it cannot be found)
 * @see                    java.util.Vector
 */
protected int indexOfRemoteObjectInVector(Object remoteObject, Vector vector) {
    int vectorSize = vector.size();
    for (int index = 0; index < vectorSize; ++index) {
        org.omg.CORBA.Object storedObject =
            (org.omg.CORBA.Object) vector.elementAt(index);
        if (storedObject._is_equivalent((org.omg.CORBA.Object) remoteObject)) {
            return index;
        }
    }
    return -1;
}

// Instance Variables
private String description;
private String function;
private String serialNumber;
private boolean isAvailable;
private Event[] events;
private Hashtable subscribers;

```

```

private Thread thread;

// Instance Initializer
{
    description = new String();
    function = new String();
    serialNumber = new String();
    isAvailable = true;
    events = new Event[0];
    subscribers = new Hashtable(10);
    thread = null;
}
}

```

## B.2 SingleResourceDemoServer.java

```

package Server;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.SystemException;
import Equip.EquipmentResource;
import Equip.EquipmentResourceImplementation;
import java.util.Vector;
import Notification.Event;

/**
 * Implements a CORBA server that exposes a single, simplified OBEM Equipment Resource.
 * It broadcasts a particular event at a regular interval. Subscribers may subscribe
 * to receive this event or any other (although they will not receive notification of
 * any other event).
 *
 * @author Gordon Davis
 * @version 0.0
 */
public class SingleResourceDemoServer {

    /**
     * Implements the server.
     *
     * @param args    arg[0] is the index into the event array of the event to be broadcast.
     *                arg[1] is the broadcast interval in milliseconds.
     * @see String
     */
    public static void main(String[] args) {

        // Parse the arguments
        if (args.length != 2) {
            System.out.println("Usage: java Server.SingleResourceDemoServer " +
                "<index of event to broadcast> <ms between broadcasts>.");
            System.exit(1);
        }
    }
}

```

```

    }

    // Create all events
    Event[] events = createEvents();

    // Validate args[0] (index of event to be broadcast) and derive
    // the proper event ID from it.
    int eventIndex = 0;
    try {eventIndex = Integer.parseInt(args[0]);}
    catch (NumberFormatException e) {}
    if (eventIndex > events.length - 1) {
        System.out.println("Event index must be less than " + events.length);
        System.exit(1);
    }
    String idOfEventToBroadcast = events[eventIndex].eventID;

    // Validate args[1] (time interval between broadcasts in ms).
    int broadcastInterval = 2000;
    try {broadcastInterval = Integer.parseInt(args[1]);}
    catch (NumberFormatException e) {}
    if (broadcastInterval < 1) {
        System.out.println("Broadcast interval must be at least 1.");
        System.exit(1);
    }

    //Initialize ORB
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();

    EquipmentResource equipResourceImpl = null;
    try {

        // Create the EquipmentResource Implementation
        equipResourceImpl =
            new EquipmentResourceImplementation(
                "New Equipment",
                "Make Something",
                "A7B85JWX15324",
                true,
                events,
                idOfEventToBroadcast,
                broadcastInterval
            );
        System.out.println("EquipSrv: impl is ready.");

        // Receive events from clients.
        _CORBA.Orbix.impl_is_ready ("EquipSrv", 120000); // time out = 2 minute
        try {Thread.sleep(120000);} catch (InterruptedException e) {}
        System.out.println("Shutting down server...");
    }
    catch (SystemException se) {
        System.err.println("Exception raised during operation of SingleResourceDemoServer" +
se.toString());
        System.exit(1);
    }
    System.out.println ("Server exiting....");
    return;

```

```

    }

    /**
     * Creates all events for the <code>EquipmentResource</code> implementation.
     *
     * @return all events
     * @see Notification.Event
     */
    protected static Event[] createEvents() {
        Event[] eventArray = new Event[5];
        eventArray[0] = new Event("NEWEQUIPNEEDSMAINT", "New Equipment needs maintenance.",
true);

        eventArray[1] = new Event("NEWEQUIPISAVAIL", "New Equipment is available.", true);
        eventArray[2] = new Event("NEWEQUIPISUNAVAIL", "New Equipment is unavailable.", true);
        eventArray[3] = new Event("NEWEQUIPWAFERENTERED", "A wafer entered New Equipment.",
true);

        eventArray[4] = new Event("NEWEQUIPWAFEREXITED", "A wafer exited New Equipment.", true);
        return eventArray;
    }
}

```

### B.3 EquipmentObserver.java

```

package Client;

import org.omg.CORBA.ORB;
import org.omg.CORBA.SystemException;
import IE.lona.OrbixWeb._CORBA;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import CosEventComm.Disconnected;
import Equip.EquipmentResource;
import Equip.EquipmentResourceHelper;
import Equip.StringParameterTooLong;
import Notification._EventSubscriberImplBase;
import Notification.Event;
import Notification.EventNotification;
import Notification.EventNotificationHelper;
import Notification.EventGeneratorPackage.EventDoesNotExist;

/**
 * <code> EquipmentObserver </code> is a demonstration client for
 * <code> SingleResourceDemoServer </code>, a CORBA server that
 * exposes a single, simplified OBEM Equipment Resource. It
 * <ul>
 * <li> gets and changes the description of the resource,
 * <li> asks the resource for the events it broadcasts,
 * <li> subscribes to one of the events,
 * <li> and prints a console message when it receives an event notification.
 * </ul>
 * <p>
 * @author      Gordon Davis
 * @version 0.0
 *
 */

```

```

public class EquipmentObserver extends _EventSubscriberImplBase {

    /**
     * Accepts an <code> Event </code> notification from an event supplier.
     * This method implements the <code> push </code> operation in
     * <code> CosEventComm::PushConsumer </code>.
     *
     * @param data    the event notification
     * @see org.omg.CORBA.Any
     * @exception CosEventComm.Disconnected
     */
    public synchronized void push(org.omg.CORBA.Any data) throws Disconnected {
        EventNotification notification = EventNotificationHelper.extract(data);
        System.out.println("Event received (" + notification.eventID + "): " +
            notification.eventText);
        return;
    }

    /**
     * This operation is in the CosEventComm interface but is not used in this implementation.
     */
    public void disconnect_push_consumer() {
        return;
    }

    /**
     * This method implements the client portion of the demo.
     *
     * @param args the command line arguments
     * @see String
     */
    public static void main(String[] args) {

        // Decode argument(s) and prepare to bind to server.
        if (args.length != 1) {
            System.err.println("USAGE: java Client.EquipmentObserver <hostName>");
            System.exit(1);
        }
        String srvHost = new String (args[0]);
        String markerServer = ".EquipSrv";
        EquipmentResource equip = null;

        // Initialize the ORB.
        ORB.init();

        try {
            // Bind to the EquipmentResource on the server.
            System.out.println("Binding to " + markerServer + " on " + srvHost);
            equip = EquipmentResourceHelper.bind( markerServer, srvHost );

            // Get and change the description attribute of the remote EquipmentResource.
            System.out.println("");
            System.out.println("Equipment Resource Description = " + equip.getDescription());
        }
    }
}

```

```

System.out.println("Changing description to 'New Contraption");
try {
    equip.setDescription("New Contraption");
}
catch (StringParameterTooLong e) {};
System.out.println("Equipment Resource Description = " + equip.getDescription());

// Get the function and serial number attributes and find out whether the
// resource is available
System.out.println("Equipment Resource Function = " + equip.getFunction());
System.out.println("Equipment Resource Serial Number = " + equip.getSerialNumber());
if (equip.isAvailable())
    System.out.println("The Equipment Resource is AVAILABLE.");
else
    System.out.println("The Equipment Resource is not AVAILABLE.");

// Get a list of events from the remote EquipmentResource.
Event[] events = equip.getEvents();
System.out.println("");
System.out.println("EquipmentResource " + equip.getDescription() + " broadcasts the
following events:");
for (int index = 0; index < events.length; ++index)
    System.out.println("Event " + index + ": " + events[index].eventID + " " +
events[index].eventText);

// Ask the user which event to subscribe to.
System.out.println("");
System.out.print("Which event do you want to subscribe to? ");
int choice = 0;
try {
    choice = readIntegerFromKeyboard();
}
catch (IOException e) {choice = 0;}
catch (NumberFormatException e) {choice = 0;}
String eventID = events[choice].eventID;

// Create an instance of this class to receive event notifications.
EquipmentObserver observer = new EquipmentObserver();

// Subscribe to the chosen event.
try {
    equip.subscribeToEvent(eventID, observer);
    System.out.println("Successfully subscribed to " + eventID);
}
catch (EventDoesNotExist e) {
    System.err.println("Tried to subscribe to event: " + eventID + " but it does not
exist.");
}

// Process incoming events (this would include all incoming messages (not just
// event notifications).
System.out.println("Processing events...");
_CORBA.Orbix.processEvents();
}
catch (SystemException ex) {
    System.err.println("Exception during bind");
}

```

```
        System.err.println(ex.toString());
        ex.printStackTrace();
        System.exit(1);
    }
    // We don't want to exit right away.
}

/**
 * Reads characters from the keyboard, interprets them as an <code> int </code>,
 * and returns the value.
 *
 * @return the <code> int </code> interpreted from the keyboard input.
 */
public static int readIntegerFromKeyboard()
    throws IOException, NumberFormatException {

    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String line = in.readLine();
    return Integer.parseInt(line);
}
}
```





**SEMATECH Technology Transfer  
2706 Montopolis Drive  
Austin, TX 78741**

**<http://www.sematech.org>  
e-mail: [info@sematech.org](mailto:info@sematech.org)**