FINE GRAINED CORBA SERVICES TO BUILD SCALABLE DRE ARCHITECTURES

Victor Giddings (Objective Interface Systems, Inc., Herndon, VA)
victor.giddings@ois.com

ABSTRACT

As well as the functional services directly provided, the CORBA Services, in particular the Naming and Events Services, provide important "architectural glue" to systems built with CORBA. Unfortunately, these services are traditionally implemented as monolithic standalone servers. This paper will discuss experiences with the use of a set of fine-grained CORBA services that are more suitable for architecting scalable Distributed Real-time and Embedded (DRE) systems. These library-based implementations are relatively "light-weight" in both CPU and memory usage, and are hosted by an RTCORBA-compliant ORB. These characteristics combine synergistically with CORBA's location transparency to provide an "architectural transparency". This provides some of the benefits, such as flexible system assembly, that are attributed to component-based software architectures without the need for additional standards and their associated costs.

1. CORBA SERVICES

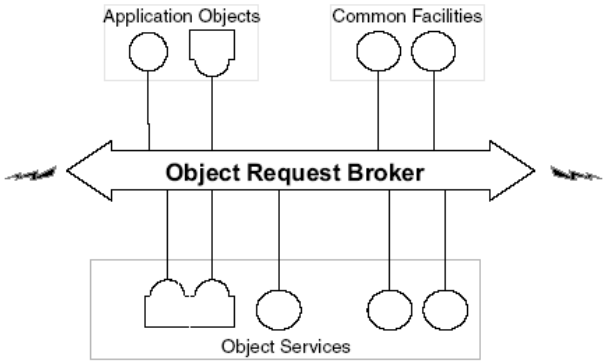

Figure 1: OMG Object Management Architecture

The above diagram shows the Object Model Architecture[1] developed by the Object Management Group as a context for the Common Object Request Broker Architecture[2]. While the focus of the architecture is the Object Request Broker, there is provision for a set of generally useful Object Services, as well as domain-crossing Common Facilities. Object Services are "a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independent of application domains. [3]"

There are a large number of defined Object Services:

- Additional Structuring Mechanisms for the OTS
- Collection Service
- Concurrency Service
- Enhanced View of Time
- Event Service
- Externalization Service
- Licensing Service
- Life Cycle Service
- Lightweight Log Service
- Management of Event Domains
- Naming Service
- Notification Service
- Persistent State Service
- Property Service
- Query Service
- Relationship Service
- Security Service
- Telecoms Log Service
- Time Service
- Trading Object Service
- Transaction Service

Of these, the Naming Service and the Event Service are most often found in embedded systems.

1.1 Naming Service

The CORBA Naming Service helps to solve the "initial reference problem" for CORBA clients, i.e., that clients must obtain an Object Reference that is generated by a CORBA server order to connect to the server to invoke operations on the objects that the server process hosts. The CORBA Naming Service server provides a "well-known place" for servers to register Object References and for clients to look them up. Servers *bind* Object References to a "Name", which consists of an arbitrary sequence of string pairs. Each string pair designates an arc that originates at a NamingContext. Sequences of these arcs result in arbitrary naming graph; this allows hierarchical naming schemes to be of arbitrary depth and accommodates aliasing. Clients *resolve* the Name to obtain the correct Object Reference. This reduces the amount of information needed to be shared between a server and its clients, and divorces the

information from the details of configuration and deployment.

The CORBA Naming Service interfaces are expressed in OMG Interface Definition Language (IDL) and implemented in CORBA servers. This moves the initial reference problem "upstream" from finding application-specific object references to finding the Object Reference for the correct NamingContext. The CORBA specification provides a special feature, the *resolve_initial_references* operation that allows the "root" NamingContext to be found. There is provision for configuring the location of the root NamingContext, in order to ensure interoperability between products and applications.

1.2 Event Service

The CORBA Event Service provides asynchronous multi-point to multi-point information transfer between *suppliers* and *consumers*, each of which may be unaware of the existence or identity of the others. An *event channel* decouples suppliers from consumers and may untyped, i.e., capable of carrying any type of information, or typed, i.e, capable of carrying a restricted set of set of information. This style of communication complements the synchronous point-to-point communication embodied in a CORBA remote invocation.

The Event Service interfaces are expressed in two sets of IDL: one that specifies the Event Channel and subsidiary objects that an Event Channel creates to connect suppliers to consumers, and one that defines the responsibilities of event suppliers and consumers. The Event Channel IDL includes different versions for the un-typed and typed Event Channel.

It should be noted that the Notification Service was defined to augment (some would say subsume) the Event Services. It adds a host of features, including event filtering, structured events (a weakly type event), and quality of service control. Most of these features are directed toward large-scale applications and not needed in DRE systems.

2. MONOLITHIC VS. FINE-GRAINED IMPLEMENTATIONS

2.1 Monolithic Implementations

Although specified solely as CORBA objects, most CORBA Service products have been implemented and delivered as monolithic, standalone servers, i.e., as separate programs that run in their own process.

There are several good reasons why this is so: Delivery as a program presents the user with a self-contained and complete implementation in a single package, thus making it easier for the provider to develop and maintain, and for the user to configure.. Most implementations offer features that are not part of the CORBA services specification, but are expected by end users in an enterprise-targeted product. For example, the Naming Service specification does not the use of persistence for the recovery of the naming server from crashes. However, most commercial products will checkpoint the contents of the naming graph to a file or database to allow a new instance of a server to recover the state of a server that may have crashed or has otherwise terminated.

This practice is somewhat reinforced by the CORBA standard method of configuring initial access to services. The resolve_initial_references operation described earlier takes one of a prescribed set of strings. For example, access to the Event Service is requested by specifying the string "EventService". Only one object reference can be obtained in this manner, thus reinforcing the notion that there is only one event channel.

2.2 Fine-Grained Implementations

In contrast, all CORBA services are specified in terms of IDL interfaces to objects. In CORBA, objects are a fairly fine-grained concept, and the location of each object is transparent to the application. Indeed, this "location transparency" is one of the major architectural advantages of CORBA over previous technologies. It allows an application to be designed and built to a distributable object-oriented client-server logical architecture and bound at deployment to a distributed physical architecture. Thus it is entirely possible that the individual instances of the objects that provide one of the CORBA services could be distributed across multiple servers or even co-located in application processes.

In fact, there are several features in the defined CORBA services that acknowledge or take advantage of this ability to distribute pieces of a service across multiple servers. The CORBA Naming Service has several operations that directly support the notion of "federated" Naming Servers: a situation where the naming graph is distributed across several servers and unified by cross-server bindings. The process of connecting suppliers and consumers to the CORBA Event Service is deliberately more complicated than necessary in order to allow composing of event channels.

Most CORBA Service implementations can be built as libraries that can be instantiated within and distributed across an application's servers. The resulting fine-grained implementations provide implementations of the CORBA interfaces required for the service, but do not provide an execution context in which they process.

There are several obvious advantages to fine-grained implementation of the CORBA Services that lead to the development of the Embedded Objects Services™ in the ORB*express*™ product line.

The first advantage is that the ORB*express* product line is targeted to embedded and real-time applications. These applications are often highly resource constrained, so that dedicating a process to each service may not be possible. They may be hosted on operating systems that are not multi-processing or have a single address space, and thus dedicating a single process requires dedicating a whole processor. These systems may also not have disk storage or other persistent storage.

The second advantage is to overcome the tight binding between a service's implementation and its underlying ORB that is inherent in a monolithic implementation. In some sense, the service implementation is only as "good" as the ORB that hosts it. By building the service as a library, the various optimizations and other compilation switches can be varied consistent with those used to build the varieties of the underlying ORB. Further, the service can be integrated with features optionally provided by and configured into the underlying ORB. These include replaceable and custom transports, and the use of the RTCORBA features, such as priority propagation and banded communication. For example, it is entirely possible to embed an ORB*express* Names: Embedded Objects capability into an application that supports only shared memory and message queues, but not TCP/IP, as the means of communications. It would be impractical to build monolithic servers to cover all possible combinations of transports and ORB features.

A third advantage is the performance improvements that can be gained from "co-location optimizations". Most ORBs expedite the handling of requests to object implementations with the same process by bypassing marshalling and any communication on the network.

These advantages lead Objective Interface Systems to undertake the development of ORBexpress Names: Embedded Objects, a fine-grained CORBA Name Service and ORBexpress Events: Embedded Objects, a fine-grained CORBA Event Service (see Table 1). After development, several unobvious advantages were discovered. These implementations of the CORBA services were useful in the assembly of applications from objects in unexpected ways. In many cases, these advantages yielded some of the benefits attributed to component-based development.

Table 1: Example Sizes of Fine-Grained CORBA Services

Embedded Objects Product	Total Library Obj Code Size (kB)	
	VxWorks/PPC (.out)	Solaris 2.8/Sparc (.a)
ORB*express* Names	80	127
ORB*express* Events	115	227

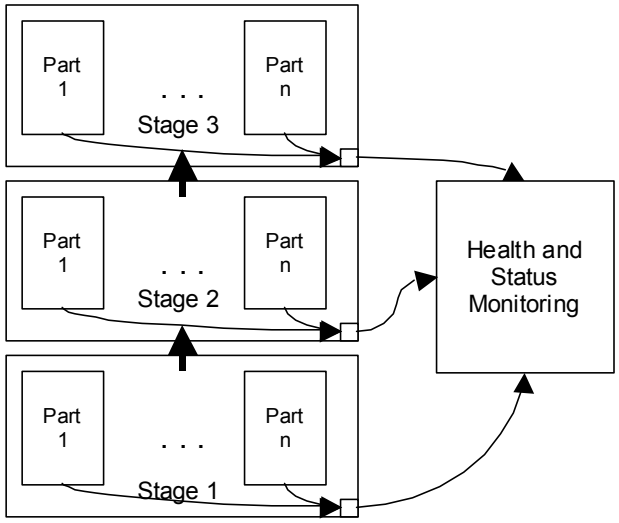

Figure 2: Functional View of Prototype Application

3. ADVANTAGES OF COMPONENT-BASED DEVELOPMENT

Component-based development has been called "the third stage of formality" [4] in software design. Components package a number of objects along with supporting services, often provided by a "framework", into a cohesive subsystem. Given sufficient reusable components and the ability to customize them, the process of application development becomes more a process of assembly, than a process of coding.

Purported technical benefits include better management of complexity that allows better quality and separation of complex infrastructure issues from the "business logic" and looser coupling among components that allows independent design and development and prevents the "rippling" effect of requirements changes. These technical advantages should result in easier, safer, and quicker development.

For these reasons, component-based development is drawing a lot of attention in the commercial development sector, as well as being a basis for the Software Communications Architecture (SCA) developed for the U.S. Army's Joint Tactical Radio System program, shaped by the Software Defined Radio Forum, and being standardized by the Object Management Group [5].

4. PROTOTYPE APPLICATION

The functional view of one prototype application that has been studied is illustrated in Figure 2. It is evocative of radar processing or other signal processing applications that must processing immense amounts of data in a pipeline

manner with each stage of the pipeline refining the results from the previous stage. Further, it is assumed that the processing at each stage may be processed in a data parallel manner. Thus the functional block at each stage contains *n* "part" processing blocks. Finally, it is assumed that the each part and each stage reports its status to a Health and Status Monitoring (H&S) subsystem.

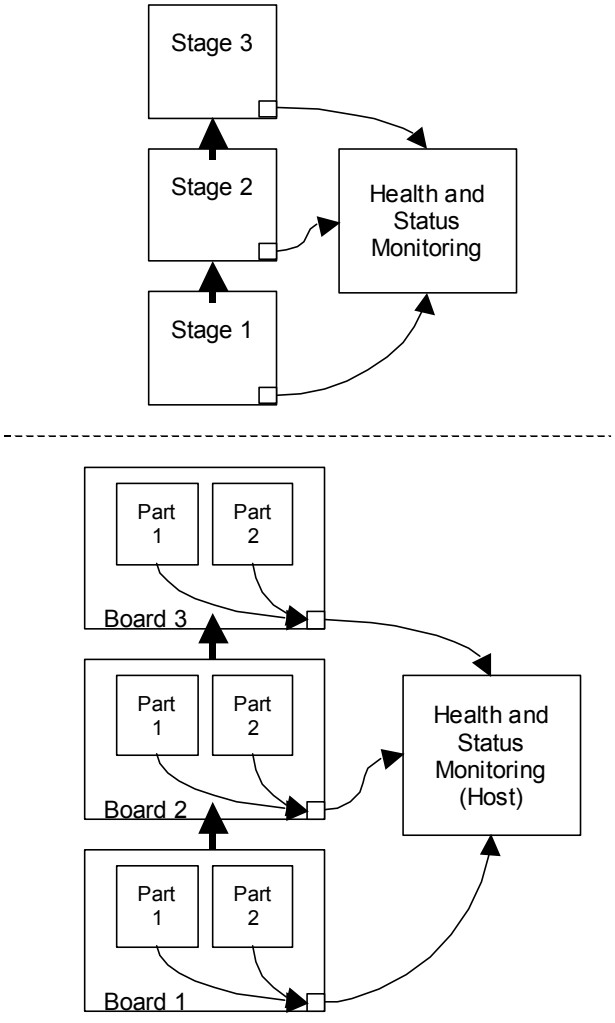

Figure 3 Two Different Hardware Architecture

The prototype application has been studied on two different hardware architectures as illustrated in Figure 3. The configuration in the top half of Figure 3 is simple; each pipeline stage is processed on a dedicated processor. The H&S subsystem is also run on a different host.

The configuration in the lower half of Figure 3 shows two part processing components sharing a board, such as a CSPI multi-computer board, and sharing the processing load for each stage of the pipeline. The H&S component remains on a separate host.

In each of these diagrams, the thicker arrows indicate CORBA method invocations, while thin arrows indicate the use of events.

The rest of this paper will illustrate the use of the fine-grained CORBA service implementations in the implementations of these different architectures, and how they allow the differences between these physical architectures to be ignored until the application is actually deployed on the target hardware.

5. USES OF FINE-GRAINED EVENT SERVICE

The CORBA Event Service has been used as an infrastructure for functions similar to the Health and Status Monitoring subsystem in many applications. The asynchronous and distributed nature of status monitoring and fault detection makes the Event Service an ideal infrastructure for this. In both of the physical architectures, the Event Service serves to collect health and status reports from each of the three processing stages. In both of these physical architectures, either a traditional, monolithic, or fine-grained implementation would be sufficient for this purpose.

In the second physical architecture, the use of the fine-grained Event Service in each board to collect events from the parts processors allows this processing to be performed without requiring a separate process on the board. Further, it allows each stage to be treated as a component regardless of whether the component is monolithic or implemented by a sub-assembly. Thus, regardless of which of the physical architectures are used; the Event Channel connected to the H&S Monitor can be configured with three suppliers in either case. Each "on-board channel" connects itself to the off-board channel as a supplier. The following code snippet illustrates this process:

```
CosEventChannelAdmin::ConsumerAdmin_var
  OnBoard_Consumer_Admin
    = OnBoard_Channel->for_consumers();
CosEventChannelAdmin::ProxyPushSupplier_var
  The_Proxy_Push_Supplier
    = The_Consumer_Admin->obtain_push_supplier();
CosEventChannelAdmin::SupplierAdmin_var
  The_Supplier_Admin
    = The_Event_Channel->for_suppliers();
CosEventChannelAdmin::ProxyPushConsumer_var
  The_Proxy_Push_Consumer
    = The_Supplier_Admin->obtain_push_consumer();
The_Proxy_Push_Supplier->connect_push_consumer
    (The_Proxy_Push_Consumer.in());
The_Proxy_Push_Consumer->connect_push_supplier
    (The_Proxy_Push_Supplier.in());
```

The fine-grained Event Service was also found to be of use in implementing the "collective request" infrastructure

```
NCONTEXT |-/Application
NCONTEXT | |-/Stage1
NOBJECT  | | |-/home
NCONTEXT | |-/Stage2
NOBJECT  | | |-/home
NCONTEXT | |-/Stage3
NOBJECT  | | |-/home
NCONTEXT | |-/H&S
NOBJECT  | | |-/home
```

Figure 4: Naming Hierarchy Relecting Top-Level Components

required by the Data Parallel CORBA specification [6]. This specification allows a single parallel request to deliver different data to parallel "parts" that process pieces of the request in parallel. If these parallel parts make a request to other parallel parts, this request is termed a *collective request*. The infrastructure supporting collective requests must synchronize the receipt of pieces of the parallel request from each of the requesting parallel parts before it can allow delivery of the data to the processing parallel parts. A fine-grained event channel was found to be of use in implementing this infrastructure. Each piece of the parallel request is pushed into the event channel, and the consumers attached to each part wait for all pieces of the request to arrive before forwarding the request to the processing pieces.

6. USE OF FINE-GRAINED NAMING SERVICE

The Naming Service was used in two different ways within the studies application. The first was a more traditional use of the Naming Services to find the known top-level components of the application. For example, Figure 4 shows the Naming Hierarchy for the example application running on the first hardware architecture:

```
NCONTEXT |-/Application
NCONTEXT | |-/Stage1
NOBJECT  | | |-/home
NOBJECT  | | |-/Part1
NOBJECT  | | |-/Part2
NOBJECT  | | |-/BoardChannel
NCONTEXT | | |-/CollectiveInvoke
NOBJECT  | | | |-/home
NOBJECT  | | | |-/RequestChannel
NOBJECT  | | | |-/PushProxy1
NOBJECT  | | | |-/PushProxy2
NCONTEXT | |-/Stage2
NOBJECT  | | |-/home
NOBJECT  | | |-/Part1
NOBJECT  | | |-/Part2
NOBJECT  | | |-/BoardChannel
NCONTEXT | | |-/CollectiveInvoke
NOBJECT  | | | |-/home
NOBJECT  | | | |-/RequestChannel
NOBJECT  | | | |-/PushProxy1
NOBJECT  | | | |-/PushProxy2
NCONTEXT | |-/Stage3
NOBJECT  | | |-/home
NOBJECT  | | |-/Part1
NOBJECT  | | |-/Part2
NOBJECT  | | |-/BoardChannel
NCONTEXT | |-/H&S
NOBJECT  | | |-/home
```

Figure 5: Naming Hierarchy Relecting Sub-assemblies

This use of the Naming Service is straightforward and application dependent. Servers register the objects that they host under application-dependent, and probably hard coded, names. Clients of these objects look them up by name.

The second use of the Naming Service supports the transparency of using sub-assemblies as components, and is reflected in Figure 5:

The Naming Service is used here in different way. Instead of being used by the application directly, it is used to inform elements of subassemblies where other elements of the subassembly are. Part of this processing is again hard-coded; for example, the "stage" subassemblies know there is one board-specific event channel named "BoardChannel". But, there is also part of the process that requires reflection. For example, the number of parts processors can be varied at each stage. The configuration of the CollectiveInvoke sub-components must know the multiplicity and identity of the parts processors in the following stage. This requires the CollectiveInvoke component to discover the number and identity of the parts processors in the following stage. Fortunately, the CORBA Naming Service includes the ability to dynamically traverse the naming graph. When combined with certain naming conventions, the required process of discovery can be easily accomplished.

7. ARCHITECTURAL GLUE

These explorations have revealed unexpected uses for fine-grained implementations of these CORBA services. Once these services are freed from monolithic implementations as standalone servers, they can be used "architectural glue"; they are very useful in assembling and tying together systems from components. In the prototype application discussed above, the fine-grained Event Service was used in the following ways:

- "chained event channels" were used to collect information from sub-components without the need of the overall Health and Status Monitoring to be aware of the details of assembly
- "local event channels" were used to provide data transfer as part of implementing collective innovations

The fine-grained Naming Service was used to reflect:

- Application structure
- Application deployment
- Sub-component structure and implementation

The use of a simple naming convention allowed the built-in capabilities of the Naming Service to provide reflection capabilities to the architecture to allow dynamic query and discovery of components and sub-assemblies.

8. FUTURE DIRECTIONS

The novel uses of these services as architectural glue would be furthered if several issues were addressed in the CORBA standards.

The first of these are “factory interfaces”. There is no standard way to create a Naming Service NamingContext object except from another NamingContext. There is also no standard interface for creating EventChannels. These are addressed in the ORB*express* Embedded Object Services products as a proprietary extension.

There is also a need for location controls on some of the object instances that are created by services. For example, there is a need to create the Event Service proxy objects collocated with the consumers and suppliers that will connect to them, especially if an implementation is to use multicast communication between suppliers and consumers.

There are several CORBA standards in process that are relevant to these explorations. The Lightweight CORBA Component specification may provide a conceptual framework in which the techniques explored here could be considered in an implementation of rather than a competitor to. The Deployment and Configuration specification was recently adopted and contains a clean framework to specify the configuration and deployment of CORBA-based components.

REFERENCES

1 Soley, R. M. and C. M. Stone, *Object Management Architecture Guide, Revision 3.0*, John Wiley and Sons, New York

2 OMG, *The Common Request Broker: Architecture and Specification*, OMG specification *formal/02-06-01*, Object Management Group, Natick, MA

3 OMG, *Naming Service Specification, Revised Edition, February 2001*, OMG specification *formal/01-02-65*, Object Management Group, Natick,.MA

4 Component Group, “Component-Based Development – an Overview”, http://www.componentgroup.com/whitepapers/overview.html, The Component Group, 2000

5 *Software Communications Architecture Specification*, MSRC-5000SCA, V2.2, Modular Software-programmable Radio Consortium, November 17, 2001, available at http://jtrs.army.mil/pages/sections/technicalinformation/technical_SCACurrent.html

6 OMG, *Data Parallel CORBA*, OMG specification *ptc/03-03-05*, Object Management Group, Natick, MA