ORB Middleware Evolution for Networked Embedded Systems *

Christopher Gill, Venkita Subramonian
Jeff Parsons, Huang-Ming Huang
Stephen Torri
{cdgill,venkita,parsons,hh1,storri}@cs.wustl.edu
Department of Computer Science
Washington University, St.Louis

Douglas Niehaus
niehaus@eecs.ku.edu
EECS Department
University of Kansas, Lawrence

Douglas Stuart
douglas.a.stuart@boeing.com
The Boeing Company
St. Louis. MO

Abstract

Standards-based COTS middleware has been shown to be effective in meeting a range of functional and QoS requirements for distributed real-time and embedded (DRE) systems. Each standard makes limiting assumptions, often implicit, about the fundamental set of system capabilities and constraints typical of the domain to which the standard applies. When the characteristics of a particular class of systems violates a standard's assumptions, it may be appropriate to modify or extend the standard and its conforming implementations to better match the actual characteristics of that class of systems while still exploiting the capabilities of the standard.

In this paper, we argue that key assumptions upon which even the more advanced middleware standards are based, e.g., Real-Time CORBA (RT-CORBA), are violated by an important class of DRE systems characterized by the following properties: (1) highly connected networks of (2) numerous memory-constrained endsystems, with (3) stringent timeliness requirements, and (4) support for adaptive reconfiguration of computation and communication elements, and their associated timeliness requirements. We describe our recent work on nORB, a small footprint ORB middleware framework for the Boeing Open Experimental Platform (OEP) under the DARPA NEST program, to meet this entire set of requirements by adapting, unifying, and extending patterns and techniques from earlier related research on COTS middleware frameworks, such as UBI-core, ACE, Kokyu, and TAO.

Keywords: Distributed Embedded Systems, Sensor-Actuator Networks, Real-Time Middleware.

*This work was funded in part by the DARPA NEST program. We also wish to acknowledge and thank Boeing engineers Jeanna Gossett and Tom Corcoran for their contributions to this research.

1 Introduction

A primary goal of the DARPA Networked Embedded Systems Technology (NEST) program is to enable *"fine-grain" fusion of physical and information processes* [1]. To meet this goal, a new class of Distributed Real-time and Embedded (DRE) systems, which we term *NEST-class*, is needed. The hardware infrastructure for NEST-class systems consists of a network of 100 to 10,000 fine grained computing nodes, each closely coupled with local sensors and actuators. In this paper we focus on middleware support issues for the Boeing NEST Open Experimental Platform (OEP) and its challenge problem of vibration sensing and damping in aerospace applications.

Overview of the Boeing NEST OEP: Within the broader category of NEST-class systems, a variety of endsystem, operating system, middleware, and networking infrastructures are suitable for different problem domains. For example, the Boeing NEST OEP is designed for Micro Electro-Mechanical Sensor (MEMS) vibration sensors and actuators coupled to small-scale common-of-the-shelf (COTS) endsystems running a COTS operating system, and communicating via a small-scale middleware framework over wired network connections.

From the middleware perspective, sensor-actuator-processor meshes have two characteristics of particular interest: incremental data diffusion and remote interaction. Requirements arising from both of these characteristics combine to motivate our choice of the kind of middleware we are applying to the Boeing NEST OEP:

- To manage sensor/actuator behavior effectively, the software must exhibit *real-time* behavior at an appropriate temporal scale relative to the physical properties of the system.
- To analyze and coordinate remote interactions within

Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE

those time scales, *remote method invocation* is favored over a diffusing computation approach.

- To support software engineering and re-use goals of the overall project, we specifically employ a distributed object computing (DOC) style of remote method invocation.

Challenges for Middleware in the Boeing NEST OEP: The key challenges for middleware design and implementation in the Boeing NEST OEP are to: 1) provide a robust and portable DOC middleware framework, 2) re-use existing infrastructure, patterns, and techniques, 3) enforce real-time middleware QoS assurances, 4) support time-bounded adaptation of middleware QoS properties, 5) reduce middleware footprint, and 6) interoperate with standards-based middleware.

The rest of this paper is structured as follows. Section 2 summarizes other middleware research related to the work described in this paper. Section 3 outlines alternative approaches to evolving ORB middleware to meet these challenges, while preserving the value of existing solutions. Section 4 describes in detail the design and implementation challenges addressed as we progress towards a complete ORB solution for the Boeing NEST OEP. Finally, Section 5 summarizes our observations in this work, and offers concluding remarks.

2 Related Work

ACE: The ADAPTIVE Communication Environment (ACE) framework [2, 3] addresses the challenge of providing a robust and portable DOC middleware substrate. ACE reduces the complexity of the programming model for writing distributed OO applications and middleware infrastructure, and meets the challenge of reuse [4] of infrastructure, patterns, and techniques.

Kokyu: Kokyu [5] is a low-level middleware framework built on ACE, for flexible multi-paradigm scheduling [6] and configurable dispatching of real-time operations. Kokyu abstracts combinations of fundamental mechanisms to enforce a variety of real-time policies, including well-known strategies such as Rate-Monotonic Scheduling (RMS) [7], Earliest Deadline First (EDF) [7], and Maximum Urgency First (MUF) [8].

TAO: TAO [9, 10] is a widely used standards-compliant (ORB) built using the ACE framework. Footprint reduction has been addressed in TAO through feature subsetting beyond the level of the Minimum CORBA standard [11]. However, as we discuss in Section 3, the level of subsetting in TAO motivates our choice of a bottom-up compositional approach starting from the ACE level.

MicroQoSCORBA: MicroQoSCORBA [12] is a middleware research project at Washington State University, focusing on extreme middleware footprint reduction [13] through customization [14] of middleware features for deeply embedded systems.

Ubiquitous CORBA: Ubiquitous CORBA projects such as LegORB [15] and the CORBA specialization [16] of the minimal Universally Interoperable Core (UIC) [17] focus on a metaprogramming framework approach to DOC middleware.

3 Approaches to ORB Evolution

Two key design tensions are played out among the various approaches to DOC middleware described in Section 2. The first tension is whether the resulting infrastructure is *generated* directly from primitive elements, or *completed* using the support offered by a framework. The second tension is between complexity and flexibility in choosing exactly which features are present in middleware support tailored to a particular application domain.

In this section we examine both of these tensions, and present a rationale for our ORB solution described in Section 4. Section 3.1 compares and contrasts distinct approaches to infrastructure composition. Section 3.2 identifies specific middleware features and patterns of interest from selected examples among the related projects described in Section 2 that we have applied in developing our solution described in Section 4. Throughout this section, we consider the issue of how to achieve re-use while evolving middleware infrastructure for new problem domains and still preserving the value of existing solutions.

3.1 Composition Approaches

Several approaches are possible to compose an application from primitive elements such as those ACE provides.

Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE

First, an application may be composed directly from a collection of primitive elements, as for applications built directly from ACE or Kokyu.

To constrain the potential complexity of generating applications directly from primitive elements, a second approach is to capture the known structure of an application domain within a framework, *e.g.*, TAO. If a new application from that domain is required, a third approach is to configure the framework to form a complete application, *e.g.*, configuring TAO strategies via command-line options. A fourth approach is needed for the case where a new application is required, but the application is outside the domain for which an existing framework was designed. If the differences between the framework and the desired application are manageable, the framework can be refactored and then configured to instantiate the application, *e.g.*, as with selecting features in MicroQoSCORBA.

Fifth, if the target application is known to belong to a domain from which additional applications are desired, the framework itself may be refactored with respect to the more general criteria of the new domain. Ideally, this is done by further extraction of a common meta-level framework, as is done in Ubiquitous CORBA. This approach results in a new reusable framework framework. Our solution is to apply the fifth kind of transformation to existing frameworks whose assumptions are close to, but not the same as, those of the domain of NEST-class systems. We now relate each of the frameworks we have reused in our solution approach described in section 4.

3.2 Features and Patterns of Interest

Of the approaches discussed in Section 3.1, we have selected the ACE, Kokyu, TAO, and Ubiquitous CORBA approaches as most relevant to designing middleware for the Boeing NEST OEP. Figure 1 illustrates the re-use relationships between the ACE, Kokyu, TAO, Ubiquitous CORBA, and nORB software frameworks. ACE elements are used by Kokyu, TAO, and nORB, and the patterns reified within ACE are also evident in Ubiquitous CORBA. Patterns and elements of Kokyu have been reused within the TAO Event Service and the nORB framework for real-time scheduling and dispatching. Finally, patterns from both TAO and Ubiquitous CORBA have been re-used in the nORB framework. We now consider the levels of reuse achieved from each of these related projects.

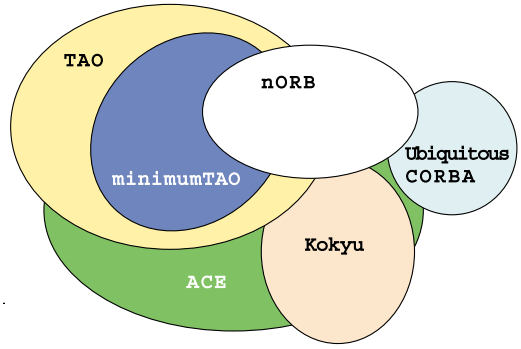

Figure 1: Intersection of Feature Sets

Re-use from ACE: Our solution leverages ACE as a suitable collection of primitives from which to build a robust and portable middleware infrastructure. Because of the reasonably fine granularity of the abstractions in ACE, we can meet the functional requirements of an ORB while retaining control over footprint and timeliness properties.

Re-use from Kokyu: Kokyu supports strategized composition of thread, timer, and queueing primitives to generate different kinds of QoS enforcement lanes (a generalization of the RT-CORBA concept of priority lanes). Developers using Kokyu can implement multiple scheduling paradigms [6] to enforce a range of real-time requirements in DRE systems. Our solution leverages Kokyu's flexibility and simplicity to compose these kinds of primitives, to provide effective enforcement of real-time constraints while minimizing footprint.

Re-use from TAO: Our solution adopts a number of ORB middleware patterns and pattern languages from TAO and its services, including pluggable protocols and request dispatching idioms and patterns. We did not, however, find opportunities for significant direct re-use of TAO infrastructure, due to two factors. First, the TAO infrastructure contains numerous coupled features and strategies resulting in a web of dependence that proved difficult to decouple for our narrower focus and stringent footprint requirements. Second, many of the features we needed were available in ACE, due in large part to previous efforts to push key general-purpose abstractions down from the TAO architectural level to the ACE level. For example, request marshalling classes had been factored out of TAO and added to ACE for use in general-purpose network communication handlers outside of TAO.

Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE

Re-use from Ubiquitous CORBA: Interestingly, the implementation of Ubiquitous CORBA did not offer as great an opportunity for re-use as we had envisioned in our earliest work on the nORB framework. Fundamentally, the tension is between the elegance of a pure abstraction based on the patterns for middleware, and the compromises needed to achieve a robust and portable solution across operating systems. ACE ultimately offered a more robust substrate for our work, and better fit our objectives of providing industrial-grade ORB middleware. We did leverage a number of the design ideas from UIC-CORBA in particular, especially in the selection of which ORB features were essential, and the structure of the object-adapter discussed in Section 4.3.

4 ORB Solution

In this section we describe our midleware solution, an ORB framework we call *nORB*[1]. We designed nORB by selecting the best patterns and techniques, and where possible infrastructure, from each of the projects described in Section 3.2. We then added extensions to meet the entire set of challenges outlined in Section 1.

The remainder of this section focuses on the technical details of our primary areas of work, highlighting elements we have been able to re-use, along with necessary extensions, modifications, and exclusions. Section 4.1 first describes how we have taken a similar approach to MicroQoSCORBA in selectively incorporating features of the GIOP protocol. Section 4.2 discusses other messaging issues such as pluggable transports, environment-specific inter-ORB protocols (ESIOPs) and object references. Section 4.3 considers different approaches to server-side object adapters, and describes how we have reconciled the alternative approaches offered by Ubiquitous CORBA and TAO. Section 4.4 describes our work on both the generated code and the compiler for the CORBA Interface Definition Language (IDL). Finally, Section 4.5 examines time-bounded priority adaptation and describes how we have modified the TAO extensions to RT-CORBA for client-side priority-connection policies, to support transparent priority reassignment.

[1] "nORB" is derived from both "NEST ORB" in honor of the domain that motivated this research, and "nano ORB" as an ironic reference to an idealized footprint size, *as long as all other design criteria are met.*

4.1 GIOP Specification

nORB supports the spirit of the GIOP specification via: a reliable connection-oriented byte stream , *i.e.*, one that delivers bytes in order and exaxtly once, and a transport that notifies connection endpoints of an unexpected connection loss. However, we have omitted or modified some GIOP features, to reduce footprint while retaining features essential to the requirements of NEST-class systems.

GIOP Features: nORB supports four of the eight GIOP 1.0 message types: Request, Reply, LocateRequest and LocateReply. We expect that connection lifetimes will often closely match the ORB's lifetime, and that requests will be fairly homogeneous and often sent repeatedly at regular intervals. These conditions reduce the usefulness of CancelRequest and CloseConnection. Our assumption of homogeneity among nodes eliminates the need for MessageErrors, used to signal discovery of an unknown message type or GIOP version. GIOP 1.0 does not recognize the Fragment message type, so it has been eliminated as well. LocateRequest and LocateReply can be used to resolve any ORB services that may be available. Since objects in NEST-class environments are often bound to particular endsystems, we do not support object migration in nORB. Therefore, our enumeration of LocateStatusType does not contain `OBJECT_FORWARD`, only `OBJECT_HERE` and `UNKNOWN_OBJECT`. Similarly, we do not support connection multiplexing or overlapping or asynchronous, requests.

GIOP Format: The GIOP header and request headers for the features implemented have been reproduced faithfully from the GIOP 1.0 specification. It has been shown [18] that eliminating bytes from these headers, *i.e.*, in the GIOPLite protocol mentioned in Section 4.2, achieves in the common case at most a few percent reduction in request size. The CORBA CDR wire protocol format is similarly efficient for marshaling basic data types, and it adds minimal complexity and overhead to the marshaling of aggregate types. It is difficult to improve upon, even for specialized domains, and we made no attempt to do so in nORB. CDR representations on the wire are fully compliant for every data type supported. the unsupported types include fixed-point decimals, abstract interfaces, value types, and the CORBA pseudo objects – TypeCode, Principal, and Context.

Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE

Exception Support: All exceptions in nORB are instantiations of a single class, NORB::Exception, which is marshaled as two two-byte fields. We can thus encode all CORBA system exceptions by their defined major and minor codes. We also reserve value ranges for application-defined user exceptions. nORB thus reduces footprint compared to a conventional CORBA implementation, at a small cost of extensibility for user-level exceptions.

4.2 Transports, Protocols, and References

In addition to the GIOP specification described in Section 4.1, we address three other topics related to method invocation in the CORBA standard. These are:

Pluggable Transports: Like TAO, nORB is capable of supporting multiple transports, since it is built on top of the ACE [2, 19] framework. ACE implements the Acceptor, Connector and Reactor patterns [20] in a way that is only loosely coupled to the underlying transport layer.

Pluggable Protocols: The pluggable transport architecture may be extended to the messaging protocol layer to support non-GIOP ESIOPs, as is done in both nORB and TAO. Since nORB only supports a subset of the GIOP messaging layer features, as described in Section 4.1, our solution could be considered a kind of ESIOP, although it still assumes the existence of a connection oriented protocol. Similarly, TAO supports an ESIOP called *GIOPLite* [18] that supports all GIOP message types, but removes certain fields of the GIOP header.

Reference Format: The nORB framework is faithful to the Limited-Profile conformance specification for Interoperable Object References (IORs). This means that although an IOR with multiple profiles will be accepted by the nORB framework, only one protocol is implemented in nORB itself (in this case IIOP 1.0) and only information from the supported protocol will be retained in an object reference derived from the one received. No context information, whether operation-related or service-related is supported in a nORB framework IOR, and of the standard CORBA operations for object references, `_is_a`, `_non_existent`, `_interface`, and `_domain_managers`, only `_is_a` is supported by nORB, as it is essential to the narrowing process.

4.3 Object Adapters

In standard CORBA, each server-side ORB may provide multiple object adapters [21]. Servant objects register with an object adapter, which demultiplexes each client request to the appropriate servant. Each object adapter may be associated with a set of policies, *e.g.*, for threading, retention and lifespan [22]. In TAO, as in all compliant implementations of recent CORBA specifications, multiple object adapters are supported by each ORB. This allows heterogeneous object policies to be implemented in a client-server environment, which is desirable in applications such as on-line banking, where each object on a server may be configured according to preferences of the server administrator, or even the end user.

In the nORB and Ubiquitous CORBA approaches, however, there is no assumption of multiple object adapters. Instead, a single object adapter per ORB is considered preferable for simplicity and footprint reduction. In nORB, the number of objects hosted on a tiny node is expected to be very small, which reduces the need for multiple policies and thus for multiple object adapters.

4.4 IDL Compilation

Interface Definition Language (IDL) offers a platform and language independent interface specification capability for ORB middleware. As in other areas, our solution seeks to adhere to the spirit of the CORBA standard, while departing from the standard as necessary to meet the stringent footprint and timeliness requirements of NEST-class systems. We describe two main areas of work on IDL compilation for nORB. First, Section 4.4.1 describes changes to the servant skeletons generated by the IDL compiler to remove virtual functions. Second, Section 4.4.2 discusses modifications to the IDL compiler itself to remove capabilities not needed for NEST.

4.4.1 IDL Skeletons

IDL-generated skeleton classes are responsible for marshalling and demarshalling parameters and return values of interface methods and dispatching the remote calls to the appropriate servant object. In the standard CORBA C++ mapping, skeleton classes 1) contain pure virtual functions for the IDL interface and mar-

Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE

shalling/demarshalling functions, and 2) use virtual inheritance to ensure correct behavior. There are several problems with this approach, for NEST-class systems. First, virtual inheritance increases footprint in the single inheritance case, by requiring spurious storage of a virtual base class pointer in the object. Second, virtual functions may inhibit inlining the operation implementations. Third, the presence of virtual functions prevents safe in-memory access by multiple processes to servants residing in shared memory, bypassing the transport layer. We make several changes to the IDL skeletons, to address these problems.

Virtual Functions and Inheritance: In many NEST-class systems, each object is likely to implement a single interface and thus inherit from a single skeleton base class. We therefore adopt the "curiously recurring template" pattern [23] to eliminate virtual inheritance and virtual functions in the skeletons. The skeleton classes become template classes that take the implementation classes as template parameters. In addition, the IDL compiler generates a tie class for binding the skeleton classes and the implementation class. Our approach replaces each virtual function call with a `this` pointer adjustment and a non-virtual function call, which is slightly more efficient.

Although we remove virtual functions and virtual inheritance from the skeletons and thus the servants inheriting from them, one virtual function used by the transport layer for dispatching remote calls was unavoidable. Without virtual function polymorphism, the *nORB* transport layer could only dispatch upcalls to a single servant class, which would obviate the benefits of IDL. Fortunately, that virtual function is only accessed by the server process during a full remote invocation, and thus does not impact local shared-memory access to the servant.

Avoiding Code Bloat: A potential drawback is that our approach could lead to code and footprint bloat, if there were more than one *nORB* object implementing the same interface on a given ORB. Our solution is to allow the application to choose whether to bind those interface operations statically or dynamically in the skeleton classes. In addition, the IDL compiler will generate abstract classes for the interfaces as well as default binding traits.

IDL Sequences: Finally, we provide a new implementation of sequences in nORB. In TAO, each IDL sequence generates its own set of classes. In nORB, sequences are mapped to the `ACE_Vector` class template. We provide generalized and specialized versions of the CDR insertion and extraction operators for `ACE_Vector`. For example, there is a specialized version of the CDR insertion and extraction operators for `ACE_Vector<Octet>`, which takes advantage of the internal representation of `ACE_Vector` being contiguous in memory.

4.4.2 IDL Compiler Refactoring

The design of the nORB IDL compiler is not only based on the TAO IDL compiler, but reuses some of its parts. The TAO IDL compiler implementation was originally monolithic but has recently been made modular, consisting of a front-end (FE) library, a pluggable back-end (BE) library and a top-level executive.

Direct Re-use: We re-use the FE library as is, since its function is independent of code generation. Furthermore, the FE parsing engine is robust, following years of real-world testing and debugging in an open-source setting. We only made slight modifications to the top-level executive, to change the set of command line options.

Refactoring: For the BE library, however, more significant changes were appropriate. First, stub and skeleton code generated from IDL was often tightly coupled to TAO internals to optimize performance and code reuse. Second, it was not so obvious initially to what extent the BE library would be reusable for nORB. However, previous refactoring of the TAO IDL compiler BE library, using design patterns such as Visitor and Factory Method, enabled a relatively straightforward retargeting of the BE library for the nORB IDL compiler, with significant reuse *of the portions of the BE library relevant to nORB*.

Reducing Compiler Complexity: Because the nORB IDL compiler does not support Anys, AMI or OBV, nor generate separate files for inlining and template classes, it generates only 4 files and makes only 6 passes over the AST, compared to 9 files and 9 to 19 passes for the TAO IDL compiler. Finally, excluding support for these features, as well as for portable interceptors, collocation optimizations and smart proxies (all supported by TAO), allows the nORB IDL compiler to use less than one-third as many visitor classes as the TAO IDL compiler.

Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE

4.5 Priority Adaptation

To meet critical timeliness requirements in many DRE systems, a priority is associated with every task. Many canonical DRE systems have periodic tasks that run at fixed priorities, which do not change over time. However, this assumption may not hold for NEST-class systems. For example, in the Boeing NEST OEP different nodes may participate in damping different vibration modes over time. Therefore, the nORB framework must support adaptive changes to task priorities. In this section, we discuss our approach to priority adaption as it relates to RT-CORBA in general and to TAO in particular.

RT-CORBA and TAO extensions: To achieve timeliness assurances in priority-based systems, the priority of a remote request must be preserved end-to-end. Due to heterogeneity of operating system thread priority schemes, a single priority value could be interpreted differently by different operating systems. RT-CORBA [24] addresses this problem by defining a uniform higher-level *CORBA Priority* that is mapped to particular end-system thread priorities. RT-CORBA uses two priority propagation models [25] - Client Propagated and Server Declared - for end-to-end priority propagation.

A key issue is whether or not the IOR maintains priority state. In the case of the Server Declared Priority Model, the priority of the server is maintained by the IOR, which is therefore stateful. The IOR uses a pre-established connection to the server, which may dedicate one or more I/O threads to serve requests at the appropriate priority. In the Client Propagated Priority Model, the priority state is instead maintained by the application, and is propagated to the server as part of the remote invocation message.

There are potential problems associated with mechanisms used to achieve these models. In particular, RT-CORBA does not specify the priority at which the I/O thread (which reads incoming requests) should be run on the server. This could lead to unbounded priority inversions [18]. TAO extends RT-CORBA by providing prioritized connection endpoints [25], where there is a $(endpoint, priority)$ mapping on the server. An IOR for a multi-priority object contains multiple profiles, each with a $(endpoint, priority)$ pair. The client ORB chooses the appropriate profile based on a priority policy with which the ORB is configured.

Priority propagation in nORB: nORB closely follows the dispatching infrastructure used in Kokyu [5, 6] and the end-to-end priority propagation mechannisms used in TAO. A restricted set of policies is used in the client ORB. The client ORB uses the priority of the thread in which the remote invocation is made. nORB then looks for a matching priority from the set of profiles in the IOR and then makes a connection to the appropriate port. We use a cached connection strategy [26] to avoid the overhead of a connection setup everytime a remote invocation is made. To alleviate priority inversion, each connection endpoint on the server is associated to a thread/reactor pair. Following RT-CORBA, we call this $(endpoint, thread, reactor)$ tuple a *dispatching lane*. The priority of each lane's thread is set so that a request in a higher priority lane will be processed before all requests in lower priority lanes.

IOR Priority Retargeting: Even though the extensions described above address some challenges for priority-based real-time applications, they do not address challenges related to adaptive reconfiguration of priorities at run-time. For example, the CORBA Server Declared Priority Model assumes the priority of the server remains fixed. However, if the priority of the server were changed at run time, then the priority information carried in the IOR would no longer be valid.

Problem: To update the priority of an object, a state change would need to be propagated to all IORs referring to that object. In a CORBA environment, this translates to publishing a new IOR to all clients. However, in an environment like the Boeing NEST OEP, these approaches may not scale due to constraints on processing resources, network bandwidth, and the expected number of references to each object. The alternative, Client Specified priorities, is equally unsuitable, as it would require similar arbitration of client priority decisions for assurance of feasibility end-to-end.

Solution: Storing priorities in the IOR is the major issue making Server Specified priority adaption difficult. If we add a level of indirection to the priority, we can use the same IOR that was exported originally. Instead of having a priority stored in the table, we would store a opaque id. This opaque id would uniquely identify a schedulable entity across all endsystems. For ex-

Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE

ample, in the Boeing NEST OEP each schedulable entity is identified using a global id which is a tuple - $(node\_number, component\_type, component\_id)$.

5 Concluding Remarks

In general, we have found that a patterns-based approach to software architecture, combined with careful examination of the domain-specific design forces, has allowed significant re-use of infrastructure whose own design reflects a pattern-aware approach. By transforming frameworks whose design lends itself readily to refactoring, the stringent constraints of particular embedded systems domains can be met without re-inventing major segments of software infrastructure unnecessarily.

References

[1] DARPA IXO, "Networked Embedded Software Technology (NEST)." `http://www.darpa.mil/ixo/`.

[2] D. C. Schmidt, "ACE: an Object-Oriented Framework for Developing Distributed Applications," in *Proceedings of the 6^{th} USENIX C++ Technical Conference*, (Cambridge, Massachusetts), USENIX Association, Apr. 1994.

[3] Center for Distributed Object Computing, "The ADAPTIVE Communication Environment (ACE)." www.cs.wustl.edu/∼schmidt/ACE.html, Washington University.

[4] D. C. Schmidt, "An Architectural Overview of the ACE Framework: A Case-study of Successful Cross-platform Systems Software Reuse," *;login:*, Nov. 1998.

[5] C. D. Gill, R. Cytron, and D. C. Schmidt, "Middleware Scheduling Optimization Techniques for Distributed Real-Time and Embedded Systems," in *Proceedings of the 7^{th} Workshop on Object-oriented Real-time Dependable Systems*, (San Diego, CA), IEEE, Jan. 2002.

[6] C. Gill, D. C. Schmidt, and R. Cytron, "Multi-Paradigm Scheduling for Distributed Real-Time Embedded Computing," *IEEE Proceedings Special Issue on Modeling and Design of Embedded Software*, Jan. 2003.

[7] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, vol. 20, pp. 46–61, Jan. 1973.

[8] D. B. Stewart and P. K. Khosla, "Real-Time Scheduling of Sensor-Based Control Systems," in *Real-Time Programming* (W. Halang and K. Ramamritham, eds.), Tarrytown, NY: Pergamon Press, 1992.

[9] D. C. S. et. al, "TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems," *IEEE Distributed Systems Online*, vol. 3, Feb. 2002.

[10] Center for Distributed Object Computing, "The ACE ORB (TAO)." www.cs.wustl.edu/∼schmidt/TAO.html, Washington University.

[11] Object Management Group, *Minimum CORBA - Joint Revised Submission*, OMG Document orbos/98-08-04 ed., Aug. 1998.

[12] David McKinnon, *et al.*, "MicroQoSCORBA." `http://microqoscorba.eecs.wsu.edu/`.

[13] A. D. McKinnon and D. Bakken and J. Shovic, "MicroQoSCORBA: A Reflective, QoS-Enabled, Configurable MicroCORBA With CASE Support," in *Proceedings of the Second Workshop on Real-time and Embedded Distributed Object Computing*, OMG, June 2001.

[14] A. D. McKinnon and O. Haugan and T. Damania and D. Bakken and J. Shovic, "MicroQoSCORBA: A QoS-Enabled, Reflective, and Configurable Middleware Framework for Embedded Systems." `http://microqoscorba.eecs.wsu.edu/MicroQoSCORBA-November2001.pdf`.

[15] M. Roman, M. D. Mickunas, F. Kon, and R. H. Campbell, "LegORB and Ubiquitous CORBA," in *Reflective Middleware Workshop*, ACM/IFIP, Apr. 2000.

[16] Manuel Roman and Roy H. Campbell and Fabio Kon, "Reflective Middleware: From Your Desk to Your Hand," *IEEE Distributed Systems Online*, vol. 2, July 2001.

[17] Manuel Roman, "Ubicore: Universally Interoperable Core." `www.ubi-core.com/Documentation/Universally_Interoperable_Core/universal%ly_interoperable_core.html`.

[18] I. Pyarali, C. O'Ryan, D. C. Schmidt, N. Wang, V. Kachroo, and A. Gokhale, "Applying Optimization Patterns to the Design of Real-time ORBs," in *Proceedings of the 5^{th} Conference on Object-Oriented Technologies and Systems*, (San Diego, CA), USENIX, May 1999.

[19] D. C. Schmidt, "The ADAPTIVE Communication Environment (ACE)." www.cs.wustl.edu/∼schmidt/ACE.html, 1997.

[20] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. New York: Wiley & Sons, 2000.

[21] I. Pyarali and D. C. Schmidt, "An Overview of the CORBA Portable Object Adapter," *ACM StandardView*, vol. 6, Mar. 1998.

[22] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*. Reading, MA: Addison-Wesley, 1999.

[23] J. Coplien, "Curiously recurring template patterns," *C++ Report*, vol. 7, pp. 40–43, Feb. 1995.

[24] Object Management Group, *Real-time CORBA Joint Revised Submission*, OMG Document orbos/99-02-12 ed., Mar. 1999.

[25] C. O'Ryan, D. C. Schmidt, F. Kuhns, M. Spivak, J. Parsons, I. Pyarali, and D. Levine, "Evaluating Policies and Mechanisms for Supporting Embedded, Real-Time Applications with CORBA 3.0," in *Proceedings of the 6^{th} IEEE Real-Time Technology and Applications Symposium*, (Washington DC), IEEE, May 2000.

[26] D. C. Schmidt and C. Cleeland, "Applying a Pattern Language to Develop Extensible ORB Middleware," in *Design Patterns in Communications* (L. Rising, ed.), Cambridge University Press, 2000.

Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE