

Resource Virtualization in Real-Time CORBA Middleware *

Christopher D. Gill
Department of Computer Science and Engineering
Washington University, St. Louis, MO, USA
cdgill@cse.wustl.edu

ABSTRACT

Middleware for parallel and distributed systems is designed to virtualize computation and communication resources so that a more abstract and consistent view of those resources is presented to the applications that use them. Providing such a consistent virtualization in distributed real-time and embedded systems becomes increasingly challenging due to application constraints such as timeliness and resource constraints such as CPU speed, power, memory, and bandwidth limitations, which also must be considered.

This paper describes several examples of real-time CORBA middleware and examines how different constraints impact the way in which resources are virtualized in each case. Particular attention is paid to which details are hidden from users of the middleware, which details are exposed in the middleware's programming model, and how the hidden and exposed details interact to shape middleware design and implementation choices.

Categories and Subject Descriptors: D.2.11 [Software Architectures]: Domain-specific architectures.

General Terms: Design, Standardization.

Keywords: Real-time Middleware, CORBA.

1. INTRODUCTION

Middleware for parallel and distributed systems is designed to shield application developers from many details of the underlying computing environment, and to expose only the particular details that are relevant to a particular programming model. For example, the Object Management Group (OMG)'s Common Object Request Broker Architecture (CORBA) [41] provides object reference and interface semantics so that method invocations between objects written in different programming languages or executing on different endsystems are performed transparently through a set of object request brokers (ORBs).

Such language and location transparency greatly eases the task of developing distributed applications, since the mechanisms for managing the complexities of concurrent computation and asynchronous communication are relegated to the ORBs, and sets of

objects can be programmed independently using the languages that are most suited to the application and/or developers' preferences. However, the virtualization of the underlying computing environment is not absolute, so that (even for functional properties such as the correct invocation of remote methods) some details unavoidably must be considered by application developers. For example, in CORBA the `NO_MEMORY` and `COMM_FAILURE` exceptions indicate failure of a method invocation due to memory exhaustion or communication failure, respectively, which may have occurred on a remote endsystem over which the calling program has no control.

Application developers may also choose which features of the middleware's programming model they use, to trade transparency for performance or other benefits. For example, CORBA offers two-way method invocation semantics which mimics a local method invocation in which the caller blocks until the called method returns. To improve performance of distributed applications, CORBA also offers asynchronous method invocation (AMI) [37] in which the calling method can continue to execute concurrently with the called method's execution. The two-way method invocation interface is more transparent as it only requires the caller to obtain a reference to the object and invoke the method, whereas with AMI the caller may also need to poll for the result from the called method, or register an asynchronous callback handler with the ORB.

Providing a consistent virtualization of the underlying computing environment becomes increasingly challenging for distributed real-time and embedded systems, in which application constraints on properties such as timing, or explicit limitations on system resources such as memory, are as important to the correct operation of an application as its functional properties. This has resulted in further expansion of the set of programming model abstractions exposed by the middleware. For example, the OMG's Real-time CORBA Specification [40] extends the CORBA programming model to include prioritization of method invocations and configuration of processing and communication control features, which are crucial to supporting statically scheduled priority based real-time systems.

Although the evolution of standards has helped to address the tension between hiding details application developers do not need to address, while exposing those they do, the number and variety of different design dimensions faced by developers of complex distributed real-time and embedded systems makes it difficult for any one specification to strike a suitable balance for all applications. This has led in turn to new system development approaches in which middleware programming models and the underlying policies and mechanisms they represent are configured using abstract (and often formal) models.

This paper examines the problem of striking a balance between hiding and exposing crucial details in middleware programming

*Supported in part by NSF CAREER award CCF-0448562.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES+ISSS'06 October 22–25, 2006, Seoul, Korea
Copyright 2006 ACM 1-59593-370-0/06/0010 ...\$5.00.

models, from the perspective of several previous real-time middleware projects. Section 2 describes the challenges that distributed real-time and embedded systems pose for the design and implementation of real-time CORBA middleware. Section 3 describes solution approaches that have been used to address the challenges presented in Section 2. Section 4 examines the relationships between hidden and exposed abstractions in each of the solution approaches described in Section 3. Section 5 surveys related work, and Section 6 offers concluding remarks.

2. CHALLENGES

This section describes several important challenges that must be addressed when developing middleware for distributed real-time and embedded systems. These challenges are listed from the most general to the most specific, with each subsequent challenge assuming and augmenting those listed previously.

Covering the common requirements: CORBA, like any standard, derives much of its benefit by codifying a set of required features, interfaces, and other details that cover the common requirements of a significant majority of the use cases in the application domain to which the standard is targeted. This coverage is essential to ensuring interoperability, reusability, portability, and other capabilities of software frameworks written in compliance with the standard, which in turn impact how productively those frameworks can be used by system developers in practice.

Supporting configurability: Even when a standard covers as large a set of common requirements as possible, there will necessarily be other requirements that are specific only to a small fraction of the use cases in the application domain. Adding those requirements to the standard may either (1) offer only a limited return on the non-trivial investment of effort required to define a detailed and consistent specification of those requirements (due to the small number of use cases impacted); or (2) even prove counter-productive overall (due to the potential of those requirements to preclude otherwise desirable solutions for the majority of use cases that do not have those requirements).

Therefore, approaches are needed that can cover those requirements without entangling them directly with the core of the standard itself. One such approach is to identify sub-domains (for example real-time, fault-tolerance, etc.) of the application domain to which a suitably related set of use cases belongs, and to provide extensions to the standard that pertain only to use cases within those sub-domains, but that may be ignored by other use cases. A second such approach is to standardize configuration capabilities through which new requirements *can* be covered, but which do not mandate any particular requirement. Finally, the benefits of both of these approaches can be realized by leveraging standardized configuration capabilities to offer common configuration specification templates for each sub-domain, which can be re-used across use-cases within the sub-domain, and then specialized and instantiated according to each use case's particular requirements.

Providing fine-grain customization: The ability to customize different configurations for different sub-domains is a necessary step towards covering the requirements of all use cases within the standard's targeted application domain. However, this alone is not sufficient because different applications may belong to different combinations of sub-domains. For example, one use case may require only fault-tolerance but not real-time behavior, while another use case may require both fault-tolerant and real-time behavior. It is therefore necessary to support customization at an even finer granularity than sub-domain-specific configurations. Ultimately it must be possible to define and customize different *combinations of requirements* flexibly to meet the needs of each use case.

Identifying and avoiding interference: The final challenge, which stems from the need to support different combinations of requirements from multiple sub-domains, is that while the requirements within each sub-domain are likely to be mutually compatible, compatibility among requirements drawn from multiple sub-domains is less certain. It is therefore necessary to identify the particular ways in which different combinations of requirements may interfere with each other (for example, real-time requirements emphasize predictability of worst-case latency — often at a cost of some increase in best case and/or average case latency), and to be able to remove or at least mitigate the consequences of that interference within the context of each particular use case.

3. SOLUTION APPROACHES

This section examines how the challenges presented in Section 2 have been addressed by several real-time CORBA middleware frameworks. The variety of these different middleware solutions, and the different ways in which each addresses the challenges presented in Section 2, sets the stage for the discussion in Section 4 of how standardization, middleware design and implementation, and modeling and analysis can be combined to make real-time CORBA middleware a highly effective infrastructure with which to develop demanding distributed real-time and embedded systems.

Covering the common cases: The ACE ORB (TAO) [24] supports common features of the distributed object computing domain, as specified by the CORBA [41] standard, including interoperable object references (IORs), client stub and server skeleton generation from interface definition language (IDL) files, and an object request brokers (ORBs) that implement method invocation semantics across endsystem and language boundaries. In addition to ORB features, TAO also implements common services specified by the CORBA standard, such as the Naming and Event services.

TAO also supports common features for the real-time sub-domain of the distributed object computing domain, according to the Real-time CORBA 1.0 specification [40], including configurable priority lanes and thread pools, and a real-time portable object adapter (POA) [42]; as well as modifications to the ORB itself to remove priority inversions and sources of timing variability [46], and support real-time I/O [31]. TAO also provides services that are either specializations for the real-time sub-domain of common services (such as TAO's Real-time Event Service [19]), or are added to address features particular to the real-time sub-domain (such as TAO's Real-time Scheduling Service [14]).

Even within a sub-domain further specialization may be appropriate, as exemplified by the distinction within the real-time sub-domain between statically scheduled (priority based) systems supported by the Real-time CORBA 1.0 specification [40] vs. dynamically scheduled (deadline, laxity, utility, etc. based) systems supported by the Real-time CORBA 1.2 (briefly called 2.0) specification [38]. TAO provides additional support for dynamically scheduled real-time systems, including dynamic and hybrid static/dynamic scheduling [11], distributable threads [12], and release guards [54].

Supporting configurability: The CORBA standard allows developers of distributed object computing systems to program at a higher level of abstraction (i.e., working with objects, references, and method invocations, rather than threads, sockets, and message formats), but with earlier versions of CORBA most of the benefit accrued to client-side programming (i.e., obtaining object references and calling methods on objects) rather than server-side programming (i.e., implementing objects and their supporting server environments). Version 3.0 of the CORBA standard [41] provided the CORBA Component Model (CCM) specification, which addresses server-side programming in distributed object computing systems.

The CCM specification decouples objects from the services that support them, groups related objects into components, and decouples components from other components through an interface abstraction called *ports* through which components communicate. Components can be assembled automatically and flexibly into applications in CCM, through XML descriptors that declare how components' ports should be connected and how components should be deployed automatically onto particular hosts.

Although the CCM specification addresses many common server-side concerns for the distributed object computing domain, it also specifies features that are irrelevant to sub-domains such as real-time. The Lightweight CCM [39] specification narrows the set of features to a common set that is suitable for distributed real-time and embedded systems.

The Component Oriented ACE ORB (CIAO) [53] is an implementation of the Lightweight CCM specification, which is built atop TAO and extends the lightweight CCM specification to allow XML-based configuration of Real-Time CORBA features in TAO, such as priorities. CIAO has been specialized further for the real-time systems sub-domain by allowing application characteristics (i.e., execution times), requirements (i.e., rate constraints), and deployment information (i.e., CPU speeds) to be provided and used in assembly and deployment [52], which allows distributed real-time applications to be assembled and deployed more effectively.

Some real-time applications (e.g., avionics mission computing systems [11, 13]), have stringent constraints not only on the timing behavior of the system at run-time, but also on system initialization and reboot times. For these applications, not only must middleware be able to configure real-time properties in the supporting ORBs, component containers, and other infrastructure, but the configuration actions themselves must be performed within bounded time frames. We have refined the temporal predictability of CIAO configuration, using efficient static assembly and deployment mechanisms that are suitable for real-time platforms on which many such applications are hosted [49].

Providing fine-grain customization: In addition to the ability to configure application-level component configurations, TAO offers a variety of strategies to provide fine-grain customization of ORB mechanisms including pluggable network protocols [30]; event demultiplexing [17]; and concurrency and message dispatching [22]. These strategies allow fine-grain tuning of TAO's performance to the particular needs of each application.

However, an important tension arises when real-time performance is required in *embedded* systems that also have significant constraints on memory and other system resources. A shared library footprint reduction tool [36], messaging protocol optimizations [18], and other techniques are part of an ongoing effort to allow system developers to choose which features of TAO to include in shared libraries and static memory images, thus reducing footprint and feature density appropriately.

Unfortunately the ability to decouple features at a very fine level of granularity is limited by dependencies between features that were introduced either accidentally during development of the middleware framework, or that were added intentionally but support only a limited set of use-cases within a particular domain or sub-domain. While it is possible to reduce coupling in the former case through careful analysis and refactoring of implementations, removing intentional dependencies (but only for use cases where they are not appropriate) is more problematic.

An alternative to decomposing a larger-scale framework such as TAO for each sub-domain (and perhaps for each use-case so that inappropriate intentional dependencies can be removed) is to compose an appropriate middleware configuration through combi-

nations of the foundational software mechanisms from which the larger-scale middleware framework is built. For example, TAO is built atop a rich set of portable system software mechanisms provided by the ACE [45] C++ framework. We used ACE classes to develop the nORB real-time small footprint ORB [50], whose features are selected for a networked embedded real-time systems sub-domain. In supporting this sub-domain, nORB omitted many features of the CORBA standard (i.e., *anys* and other data types that were superfluous), and modified the semantics of other features (i.e., simplifying the object adapter and supporting a restricted set of message types in the ORB core) to reduce footprint significantly while keeping its real-time performance comparable to (and for some key metrics like worst case response time, even improving on) TAO's real-time performance.

Identifying and avoiding interference: Model driven middleware approaches [15] are useful for selecting features for a particular domain (or even application), and configuring those features appropriately. However, supporting appropriate fine-grain customization involves not only which features are configured into the middleware, but also how they are used by the application, and the semantics of how features interact within relevant use cases. The problem of features interacting in ways that interfere with timing, footprint, or other constraints is of particular concern in real-time and embedded systems [48].

A multi-tiered approach to customization is helpful, as such interactions may cross-cut different levels of architectural abstraction in the application and its supporting middleware. First, domain-specific modeling languages (such as component modeling languages [2]) can be helpful for selecting suitable combinations of configuration options and avoiding unsuitable ones [29]. Then, model checking tools [43] can be applied to verify suitability of those configurations [20]. Executable timed models [5, 3] can be used to capture timing as well as concurrency semantics of foundational middleware building blocks [47], and to verify timing properties of real-time systems [34].

4. DISCUSSION

As Section 3 illustrates, a variety of solution approaches are needed to address all of the challenges given in Section 2. Furthermore, in addition to the particular set of interfaces and configuration options that it exposes, each middleware solution approach has its own internal semantic structure that must be taken into account. Design, verification, implementation, and validation of middleware based systems must represent and leverage this structure for successful re-use of designs, models, implementations, and verification and validation evidence. In this section we first discuss how each of the solution approaches presented in Section 3 virtualizes resources using the abstractions provided to it by lower layers of the system architecture. We then examine open problems that must be addressed to allow different virtualization techniques to be combined effectively and correctly, to increase the fidelity with which the programming model presented to system developers can express and control the issues specific to any particular application.

Resource virtualization: For the distributed object computing domain, TAO hides many low-level system software concerns such as connection establishment, concurrency, and message demultiplexing and dispatching, and exposes higher level abstractions such as location and language transparent object references and method interfaces. For the real-time sub-domain, TAO exposes more of its low-level internal semantics, such as the ability to configure thread pool sizes and priority lanes [42], according to the Real-Time CORBA specifications [40].

For the distributed object computing domain, CIAO further virtualizes system resources by hiding details of the services provided by TAO. For example, CIAO provides different kinds of ports, (i.e., facets, receptacles, event sources, and event sinks), which encapsulate the client-side and server-side roles of both method invocation and event passing styles of communication between objects. For the real-time sub-domain, CIAO also opens up internal details to reveal not only the Real-Time CORBA features exposed by TAO [53], but also details of the particular application's constraints and characteristics and of the hardware resources on which the application is deployed [52].

Because nORB [50] supports multiple sub-domains (i.e., real-time and small footprint), its approach to resource virtualization necessarily includes restricting the set of interfaces and supporting features that it exposes to include only those that are *explicitly required* and excluding other features whose inclusion would interfere with constraints on properties from any of the sub-domains involved. However, because different combinations of sub-domains may produce different patterns of feature interference, supporting such fine-grain specialization through selecting specific sets of features and interfaces *a priori* is naturally problematic.

Model-based approaches allow selective configuration of features and interfaces so that such interference can be avoided, while still presenting system developers with common and reasonably simple interfaces and configuration mechanisms. Techniques at several architectural levels, such as domain-specific modeling languages [2]) at the level of the TAO ORB, and timed models at the level of the ACE building blocks upon which TAO depends [50] allows concerns to be separated appropriately, but then analyzed and enforced across architectural layers.

Open problems: Although nORB cannot claim to be a fully compliant CORBA ORB, it supports the most essential features, and more importantly supports the features required by the sub-domain it targets. However, this raises an important question - how to support a related sub-domain with similar constraints to the one supported by nORB, but which requires one or more additional data or message types? How those types could be added to nORB for use cases that need it, but then eliminated for use cases that cannot tolerate it (or simply do not need it) is an important problem. Aspect-oriented middleware configuration techniques have been shown to be helpful [23], but further experience applying these and other generative programming techniques is needed.

In addition to the need for new tools and techniques for generative transformation (and/or synthesis) of customized middleware implementations and interfaces, additional research is needed both into how different combinations of properties (particularly from different sub-domains) can be combined correctly, in ways that can be enforced. Further work is also needed into techniques by which the correctness of different combinations can be proven or otherwise checked, automatically. Important open problems appear at the level of the properties themselves (for example, how domain-specific information can be exploited to support efficient and provable deadlock avoidance [44]), and at the level of tools such as model checkers (for example, how domain-specific state-space optimizations [47] can be re-used across use-cases in the domain).

5. RELATED WORK

Customizable Middleware: MicroQoS CORBA [1] reduces middleware footprint by *generating* customized instantiations of middleware for embedded systems. Zen [28] is a highly customizable real-time Java [4] CORBA object request broker. Ubiquitous CORBA projects such as the CORBA specialization of the minimal

Universally Interoperable Core (UIC) [35] use meta-programming to support a significant degree of middleware customization.

Real-Time Virtual Machines: The Real-Time Specification for Java (RTSJ) [4] defines scheduling and memory management features to allow real-time software to be written in Java. The jRate [7] implementation of the RTSJ provides additional extensions for memory management. Giotto and the E-Machine [21] provide an abstract infrastructure model and virtual machine for embedded control systems with hard real-time constraints.

Model Integrated Computing (MIC) [51]: MIC tools such as the Generic Modeling Environment (GME) [26] support development of domain-specific modeling and software synthesis environments. Ptolemy II [32] is a related tool set for embedded systems that provides a rich set of computation models including the Giotto model.

Model-Driven Middleware: Model-based middleware configuration environments, such as the CoSMIC [16] tool set, support integrated model-driven assembly, deployment and configuration of components atop real-time middleware. CADENA [20] is another integrated environment for building and modeling CORBA Component Model (CCM) [53] systems. The extensible Bogor [43] model checker has been applied to verification of event-channel based systems[8]. DREAM [34, 33, 10] allows DRE system designers to do model-based schedulability analysis of distributed real-time and embedded (DRE) systems. DREAM offers a computational model called the DRE semantic domain [33], which includes tasks, timers, event channels and schedulers, and is used to determine the schedulability of a given set of tasks with specified time and event based interactions.

Other Formal Techniques Applied to CORBA Middleware: [27] introduces new stereotypes in UML and describes ways to map these stereotypes to a process algebra (FSP) and then use model checking to detect deadlocks. Model checking has also been used to verify the CORBA GIOP protocol [25], and to verify CORBA based systems [9]. TRIO [6] is a formal language used to specify properties of CORBA-based distributed applications and uses a proof-based approach to verify their correctness.

6. CONCLUDING REMARKS

This paper has examined how different middleware frameworks have been designed to support particular sub-domains of the distributed object computing application domain to which the CORBA standard pertains. Common middleware abstractions such as references, objects, components, method invocations, ports, and assembly and deployment descriptors are used to virtualize lower level services provided by the operating system, such as threads, sockets, and event demultiplexors - the operating system services in turn virtualize hardware resources such as network devices, disks, memory, and CPUs. The benefits of this virtualization include increased portability across different operating system interfaces and mechanisms, reduced programming complexity as lower-level details are abstracted away, and removal of inappropriate dependencies among application components and between application components and underlying operating system services.

Adapting middleware solutions to particular sub-domains, and particularly to the intersections among sub-domains (e.g., real-time and small footprint), requires careful specialization that in some cases must sacrifice compliance with standards that govern broader domains, in order to achieve compliance with particular requirements of the relevant sub-domain(s). In such cases, construction of middleware solutions from foundational building blocks offers an important (and often complementary) alternative to decomposing

larger-scale frameworks that were designed for the more general domain within which the sub-domain is nested.

Multi-level modeling approaches appear promising to allow analysis and verification of customized systems involving both coarser-grain configuration choices and finer-grain specializations, particularly when applied to middleware frameworks that support significant configurability of all relevant system properties, at both coarse and fine granularity. However, applying multi-level modeling approaches effectively to a broad range of distributed real-time and embedded systems will require significant further research on composition of system properties (particularly across sub-domains and/or architectural layers), and on efficient and flexible mechanisms for run-time enforcement of the composed properties.

7. REFERENCES

- [1] A. D. McKinnon and D. Bakken and J. Shovic. MicroQoS-CORBA: 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.
- [2] K. Balasubramanian, J. Balasubramanian, J. Parsons, A. Gokhale, and D. C. Schmidt. A Platform-Independent Component Modeling Language for Distributed Real-time and Embedded Systems. In *Proceedings of the 11th Real-time Technology and Application Symposium (RTAS '05)*, pages 190–199, San Francisco, CA, Mar. 2005. IEEE.
- [3] G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. In *SFM*, pages 200–236, 2004.
- [4] G. Bollella, J. Gosling, B. Brosgol, P. Dibble, S. Furr, D. Hardin, and M. Turnbull. *The Real-time Specification for Java*. Addison-Wesley, 2000.
- [5] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF Toolset. In *Formal Methods for the Design of Real-time Systems*. Springer-Verlag LNCS 3185, 2004.
- [6] A. Coen-Porisini, M. Pradella, M. Rossi, and D. Mandrioli. A formal approach for designing corba-based applications. *ACM Trans. Softw. Eng. Methodol.*, 12(2):107–151, 2003.
- [7] A. Corsaro. *Techniques and Patterns for Safe and Efficient Real-Time Middleware*. PhD thesis, Department of Computer Science and Engineering, Washington University in St. Louis, Dec. 2004.
- [8] W. Deng, M. B. Dwyer, J. Hatcliff, G. Jung, Robby, and G. Singh. Model-checking Middleware-based Event-driven Real-time Embedded Software. Department of Computer Science, Technical Report SANToS-TR2003-2, Department of Computing and Information Sciences, Kansas State University, 2003.
- [9] G. Duval. Specification and verification of an object request broker. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 43–52, Washington, DC, USA, 1998. IEEE Computer Society.
- [10] Gabor Madl and Sherif Abdelwahed and Gabor Karsai. Automatic Verification of Component-Based Real-time CORBA Applications. In *The 25th IEEE Real-time Systems Symposium (RTSS'04)*, Lisbon, Portugal, Dec. 2004.
- [11] 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*, 91(1), Jan. 2003.
- [12] C. Gill, D. C. Schmidt, L. Mgeta, Y. Zhang, S. Torri, Y. Krishnamurthy, and I. Pyarali. Enhancing the Adaptivity of Distributed Real-time and Embedded Systems via QoS-enabled Dynamic Scheduling Middleware. *The Journal of the Brazilian Computer Society special issue on Adaptive Software Systems*, 2004.
- [13] C. D. Gill, J. M. Gossett, D. Corman, J. P. Loyall, R. E. Schantz, M. Atighetchi, and D. C. Schmidt. Integrated Adaptive QoS Management in Middleware: An Empirical Case Study. *Journal of Real-time Systems*, 29(2–3):101–130, 2005.
- [14] C. D. Gill, D. L. Levine, and D. C. Schmidt. The Design and Performance of a Real-time CORBA Scheduling Service. *Real-time Systems, The International Journal of Time-Critical Computing Systems, special issue on Real-time Middleware*, 20(2), Mar. 2001.
- [15] A. Gokhale, K. Balasubramanian, J. Balasubramanian, A. S. Krishna, G. T. Edwards, G. Deng, E. Turkay, J. Parsons, and D. C. Schmidt. Model Driven Middleware: A New Paradigm for Deploying and Provisioning Distributed Real-time and Embedded Applications. *The Journal of Science of Computer Programming: Special Issue on Model Driven Architecture*, 2005 (to appear).
- [16] A. Gokhale, B. Natarajan, D. C. Schmidt, A. Nechypurenko, J. Gray, N. Wang, S. Neema, T. Bapty, and J. Parsons. CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Component Middleware and Applications. In *Proceedings of the OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture*, Seattle, WA, Nov. 2002. ACM.
- [17] A. Gokhale and D. C. Schmidt. Evaluating the Performance of Demultiplexing Strategies for Real-time CORBA. In *Proceedings of GLOBECOM '97*, Phoenix, AZ, Nov. 1997. IEEE.
- [18] A. Gokhale and D. C. Schmidt. Optimizing a CORBA IOP Protocol Engine for Minimal Footprint Multimedia Systems. *Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems*, 17(9), Sept. 1999.
- [19] T. H. Harrison, D. L. Levine, and D. C. Schmidt. The Design and Performance of a Real-time CORBA Event Service. In *Proceedings of OOPSLA '97*, pages 184–199, Atlanta, GA, Oct. 1997. ACM.
- [20] J. Hatcliff, W. Deng, M. Dwyer, G. Jung, and V. Prasad. Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems. In *Proceedings of the 25th International Conference on Software Engineering*, Portland, OR, May 2003.
- [21] T. A. Henzinger and C. M. Kirsch. The embedded machine: predictable, portable real-time code. *SIGPLAN Not.*, 37(5):315–326, 2002.
- [22] J. Hu, I. Pyarali, and D. C. Schmidt. Measuring the Impact of Event Dispatching and Concurrency Models on Web Server Performance Over High-speed Networks. In *Proceedings of the 2nd Global Internet Conference*. IEEE, Nov. 1997.
- [23] F. Hunleth, R. Cytron, and C. Gill. Building Customizable Middleware using Aspect Oriented Programming. In *The OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, Tampa Bay, FL, Oct. 2001. ACM. www.cs.ubc.ca/~kdvolder/Workshops/OOPSLA2001/ASoC.html.
- [24] Institute for Software Integrated Systems. The ACE ORB (TAO). www.dre.vanderbilt.edu/TAO/, Vanderbilt University.
- [25] M. Kamel and S. Leue. Formalization and validation of the General Inter-ORB Protocol (GIOP) using PROMELA and

- SPIN. In *Int. Journal on Software Tools for Technology Transfer*. Springer-Verlag, 2000.
- [26] G. Karsai, S. Neema, A. Bakay, A. Ledeczki, F. Shi, and A. Gokhale. A Model-based Front-end to ACE/TAO: The Embedded System Modeling Language. In *Proceedings of the Second Annual TAO Workshop*, Arlington, VA, July 2002.
- [27] N. Kaveh and W. Emmerich. Validating distributed object and component designs. In *SFM*, pages 63–91, 2003.
- [28] R. Klefstad, D. C. Schmidt, and C. O’Ryan. Towards Highly Configurable Real-time Object Request Brokers. In *Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing (ISORC)*, Newport Beach, CA, Mar. 2002. IEEE/IFIP.
- [29] A. S. Krishna, E. Turkay, A. Gokhale, and D. C. Schmidt. Model-Driven Techniques for Evaluating the QoS of Middleware Configurations for DRE Systems. In *Proceedings of the 11th Real-time Technology and Application Symposium (RTAS ’05)*, pages 180–189, San Francisco, CA, Mar. 2005. IEEE.
- [30] F. Kuhns, C. O’Ryan, D. C. Schmidt, and J. Parsons. The Performance of TAO’s Pluggable Protocols Framework on High-speed Embedded Interconnects. Department of Computer Science, Technical Report WUCS-99-12, Washington University, St. Louis, 1999.
- [31] F. Kuhns, D. C. Schmidt, and D. L. Levine. The Design and Performance of a Real-time I/O Subsystem. In *Proceedings of the 5th IEEE Real-time Technology and Applications Symposium*, pages 154–163, Vancouver, British Columbia, Canada, June 1999. IEEE.
- [32] J. Liu, X. Liu, and E. A. Lee. Modeling Distributed Hybrid Systems in Ptolemy II. In *Proceedings of the American Control Conference*, June 2001.
- [33] G. Madl and S. Abdelwahed. Model-based analysis of distributed real-time embedded system composition. In *EMSOFT ’05: Proceedings of the 5th ACM international conference on Embedded software*, pages 371–374, New York, NY, USA, 2005. ACM Press.
- [34] G. Madl, S. Abdelwahed, and D. C. Schmidt. Verifying distributed real-time properties of embedded systems via graph transformations and model checking. *International Journal of Time-Critical Computing Systems*, 2005.
- [35] Manuel Roman. UbiCore: Universally Interoperable Core. www.ubi-core.com.
- [36] P. Mesnier. A Shared Library Footprint Reduction Tool. In *Proceedings of the Second Annual TAO Workshop*, Arlington, VA, July 2002.
- [37] Object Management Group. *CORBA Messaging Specification*. Object Management Group, OMG Document orbos/98-05-05 edition, May 1998.
- [38] Object Management Group. *Dynamic Scheduling Real-time CORBA 2.0 Joint Final Submission*, OMG Document orbos/2001-06-09 edition, June 2001.
- [39] Object Management Group. *Lightweight CCM RFP*, realtime/02-11-27 edition, Nov. 2002.
- [40] Object Management Group. *Real-time CORBA Specification*, OMG Document formal/02-08-02 edition, Aug. 2002.
- [41] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 3.0.2 edition, Dec. 2002.
- [42] I. Pyarali, D. C. Schmidt, and R. Cytron. Achieving End-to-End Predictability of the TAO Real-time CORBA ORB. In *8th IEEE Real-time Technology and Applications Symposium*, San Jose, Sept. 2002. IEEE.
- [43] Robby and Matthew Dwyer and John Hatcliff. Bogor: An Extensible and Highly-Modular Model Checking Framework. In *In the Proceedings of the Fourth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003)*, Helsinki, Finland, Sept. 2003. ACM.
- [44] C. Sanchez, H. B. Sipma, Z. Manna, V. Subramonian, and C. Gill. On Efficient Distributed Deadlock Avoidance for Real-time and Embedded Systems. In *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS ’06)*, Apr. 2006.
- [45] D. C. Schmidt. The ADAPTIVE Communication Environment (ACE). www.cs.wustl.edu/~schmidt/ACE.html, 1997.
- [46] D. C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale. Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers. *Journal of Real-time Systems, special issue on Real-time Computing in the Age of the Web and the Internet*, 21(2), 2001.
- [47] V. Subramonian. *Timed Automata Models for Principled Composition of Middleware*. PhD thesis, Washington University in St. Louis, Computer Science and Engineering Department Technical Report WUCSE-2006-23, May 2006.
- [48] V. Subramonian and C. Gill. A Generative Programming Framework for Adaptive Middleware. In *Hawaii International Conference on System Sciences, Software Technology Track, Adaptive and Evolvable Software Systems Minitrack, HICSS 2004*, Kona, HI, Jan. 2004. HICSS.
- [49] V. Subramonian, L.-J. Shen, C. Gill, and N. Wang. The Design and Performance of Dynamic and Static Configuration Mechanisms in Component Middleware for Distributed Real-time and Embedded Systems. In *The 25th IEEE Real-time Systems Symposium (RTSS)*, Lisbon, Portugal, Dec. 2004.
- [50] V. Subramonian, G. Xing, C. Gill, C. Lu, and R. Cytron. Middleware Specialization for Memory-Constrained Networked Embedded Systems. In *Proceedings of the 10th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS)*, Toronto, Canada, May 2004. IEEE.
- [51] J. Sztipanovits and G. Karsai. Model-Integrated Computing. *IEEE Computer*, 30(4):110–112, Apr. 1997.
- [52] N. Wang and C. Gill. Improving Real-time System Configuration via a QoS-aware CORBA Component Model. In *Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems Minitrack, HICSS 2004*, Kona, HI, Jan. 2004. HICSS.
- [53] N. Wang, C. Gill, D. C. Schmidt, and V. Subramonian. Configuring Real-time Aspects in Component Middleware. In *Lecture Notes in Computer Science: Proc. of the International Symposium on Distributed Objects and Applications (DOA’04)*, volume 3291, pages 1520–1537, Agia Napa, Cyprus, Oct. 2004. Springer-Verlag.
- [54] Y. Zhang, B. Thrall, S. Torri, C. Gill, and C. Lu. A Real-Time Performance Comparison of Distributable Threads and Event Channels. In *Proceedings of the 11th Real-time Technology and Application Symposium (RTAS ’05)*, San Francisco, CA, Mar. 2005. IEEE.