Enhancing Distributed Object Middleware Qualities

Arvind S. Krishna
Electrical Engineering & Computer Science Department
Vanderbilt University, Nashville, TN
arvindk@dre.vanderbilt.edu

Categories and Subject Descriptors

D.1.5 [**Programming Techniques**]: Object Oriented Programming—*Design Patterns*; D.2.5 [**Testing Debugging**]: Distributed Continuous Quality Assurance—*model driven development*

General Terms

Design, Performance, Experimentation, Standardization

1. RESEARCH MOTIVATION

Distributed Real-time and Embedded (DRE) systems are becoming increasingly widespread and important in a range of domains, including process automation (*e.g.*, hot rolling mills) and aviation (*e.g.*, avionics mission computing systems). As *distributed systems*, DRE systems require capabilities to manage connections and message transfers between separate machines; as *real-time systems*, DRE systems require predictable and efficient control over end-to-end system resources; and as *embedded systems*, DRE systems have weight, cost, and power constraints that limit their computing and memory resources. Designing DRE systems that (1) implement all the required capabilities, (2) are efficient and dependable, and (3) use limited computing resources is hard; building them on time and within budget is even harder. Researchers and developers of DRE systems must therefore address the following research challenges:

1. **Tedious and error-prone development** — Many DRE systems are developed using low-level languages, such as C and assembly languages, thereby increasing accidental complexities arising from complicated memory management, synchronization, etc.
2. **Configuration and tuning complexities** — It is hard to configure and tune key quality of service (QoS) properties, such as (1) pooling concurrency resources, (2) synchronizing concurrent operations, (3) enforcing sensor input and actuator output timing.
3. **Middleware customization complexities** — Operating conditions for large-scale DRE systems can change dynamically based on the domain of application, making it hard to provide customized solutions for each operating context.

The remainder of this paper describes how my research is (1) addressing the challenges outlined above and (2) being integrated and validated in the context of several representative DRE systems using middleware based on Real-time Java and C++.

OOPSLA'04, Oct. 24-28, 2004, Vancouver, British Columbia, Canada.
ACM 1-58113-833-4/04/0010.

2. TEDIOUS AND ERROR-PRONE DEVELOPMENT COMPLEXITIES

The emergences of Distributed object computing (DOC) middleware over the past decade has helped reduce the complexity of developing DRE systems. A standard DOC middleware solution for DRE applications is Real-time CORBA (RTCORBA) [6]. RTCORBA adds QoS control capabilities to improve DRE system predictability by bounding priority inversions, providing end-to-end priority enforcement, and end-to-end system resource management.

Although RTCORBA has been an OMG standard for several years, it has not been adopted universally due in part to its *steep learning curve*. Current RTCORBA implementations are mostly in C and C++. Technologies such as Java, which are easier to program and type-safe, have not proven themselves as suitable for DRE applications due to the unpredictability of garbage collection. The Real-time Specification for Java (RTSJ) [2] adds real-time extensions to the Java programming language, in particular a new memory management model that can be used in lieu of garbage collection. The Java mapping of the RTCORBA specification, however, does not yet leverage any RTSJ features. A key research challenge therefore involves the integration of RTCORBA with RTSJ to simplify the programming model for DRE applications. Another barrier to broader adoption of RTCORBA is the runtime and memory footprint overhead stemming from implementations that include all the code for supporting various ORB services, such as connection and data transfer protocols. In this case, a key research challenge is to develop optimization techniques that can reduce this overhead without unduly sacrificing functionality.

To address the challenges outlined above, my research focuses on developing higher-level programming language abstractions, including optimization strategies and patterns [7], for enhancing Real-time CORBA middleware using Java and the RTSJ. These strategies and patterns help make Java more suitable for DRE system development by enabling the application of RTSJ features (such as Scoped Memory and Real-time Threads) together with RTCORBA features (such as thread-pool lanes) to ensure the predictability of Java-based DRE applications. These patterns are also applicable to implementations designed using C++. The RTSJ features, in turn, leverage (1) algorithmic and data-structural optimizations for predictable ORB concurrency and request processing. To address the challenge of minimizing middleware footprint, my research has applied the Virtual Component Pattern [3] to factor out functionality from the ORB to reduce middleware footprint significantly.

3. CONFIGURATION AND TUNING COMPLEXITIES

DRE systems often execute on a multitude of platforms and user

contexts. To ensure that DRE applications and middleware meet their QoS requirements, they must be fine-tuned to specific platforms/-contexts by adjusting many (*i.e.*, 10's-100's) of configuration options. Time and resource constraints often limit developers to assessing the QoS of their DRE systems on very few configurations and extrapolate these to the much larger configuration space. In this context, the research challenges include developing (1) software processes to systematically and efficiently evaluate system QoS and (2) design tools to synthesize necessary artifacts, such as benchmarking code to evaluate system QoS for various configuration options.

A promising approach for addressing the problem of ensuring QoS satisfaction for DRE systems on a range of hardware, OS, and compiler platforms involves model-driven distributed continuous quality assurance (DCQA) techniques [5], which are designed to improve software quality and performance iteratively, opportunistically, efficiently, and continuously in multiple, geographically distributed locations. My research in this area focuses on (1) domain-specific modeling languages (DSMLs) and tools that allow developers to compose regression tests and benchmarking experiments to validate and tune the QoS of DRE systems by adjusting middleware configurations systematically and (2) documenting *configuration and customization* (C&C) patterns, which are tuple spaces consisting of configuration parameters along with their settings for each individual platform. C&C patterns benefit system architects and performance engineers by providing feedback that allows them to choose configurations that maximize QoS.

I am developing the Test Generation Modeling Language (TGML), which is a DSML that minimizes the cost of QA activities by capturing the customizability of middleware within models and automatically generating interface definitions, configuration files, test applications and scaffolding code from these models. The code generated using TGML can then be validated over a range of platforms using DCQA techniques to (re)validate and discover most significant configuration options impacting performance.

4. MIDDLEWARE CUSTOMIZATION COMPLEXITIES

DRE system infrastructure continues to expand in scope and capabilities as new protocols and mechanisms are defined at the network, OS, and middleware layers. For instance, middleware provides various protocol implementations tailored towards changing operation contexts, *e.g.*, the Internet Inter-ORB Protocol (IIOP) can be used for web-based communication, whereas the Stream Control Transmission Protocol (SCTP) can be used for fault-tolerant DRE applications.

To be applicable across many domains, middleware implementations are often designed to be *general-purpose* services that are pluggable and reusable. The C&C patterns described in Section 3 help configure middleware strategies by using information from a system's operational context, which can change statically and dynamically, to override certain general-purpose configurations. These patterns cannot, however, remove the footprint and performance overhead incurred by middleware designs that are layered to support pluggable context-specific implementations. In this context, the research challenges involve developing software environments that evaluate system properties (such as concurrency mechanisms and types of protocol implementation that are fixed at design and/or installation time) and use this information to customize middleware implementations and eliminate additional layers of indirection.

A promising approach for addressing middleware specialization complexities involve *partial evaluation and specialization techniques*. These techniques are an inter-procedural constant propagation technique that can improve performance and footprint by (1) tailoring services to the specific needs of software systems and (2) bypassing layers of abstraction to call directly to underlying platform protocols and mechanisms. Partial evaluation techniques can be applied to customize middleware for DRE systems at various levels, such as at assembly-, deployment-, and run-time. My research in this area focuses on refactoring middleware and services so they are amenable to automatic customization and optimization by using middleware specialization patterns [4]. These patterns are then mapped to middleware implementations via specialization tools, such as the DMS [1] toolkit used for domain specific program generation from language-independent models.

5. RESEARCH INTEGRATION AND VALIDATION

The contributions of my research can be classified hierarchically. At the base are novel optimization strategies and patterns that help enhance DRE system QoS and facilitate the application of promising new technologies (such as integrating RTSJ and RTCORBA) to DRE systems. These strategies and patterns have been validated and applied on ZEN (`www.zen.uci.edu`) which is an open-source RTCORBA ORB implemented using the RTSJ used in various R&D efforts. At the next level, my work on C&C patterns and TGML help configure and tune ORB implementations. These patterns can then be mapped to optimization strategies for ORB implementations in C++ and RTSJ. The C&C patterns have been validated and applied in the context of CIAO (`www.dre.vanderbilt.edu/CIAO`) QoS-enabled component middleware framework widely used the DRE domain. The Skoll DCQA environment (`www.cs.umd.edu/projects/skoll/`) is being used to validate C&C patterns on a range of hardware, OS, and compiler platforms. Finally, my work on partial evaluation extends the C&C patterns by customizing middleware directly via removing unnecessary layers of indirection. This work will be showcased in the context of CoSMIC (`www.dre.vanderbilt.edu/cosmic`), which is a toolsuite for composing, configuring and deploying DRE systems via DSMLs.

6. REFERENCES

[1] I. Baxter. *DMS: A Tool for Automating Software Quality Enhancement*. Semantic Designs (www.semdesigns.com), 2001.

[2] G. Bollella, J. Gosling, B. Brosgol, P. Dibble, S. Furr, D. Hardin, and M. Turnbull. *The Real-Time Specification for Java*. Addison-Wesley, 2000.

[3] A. Corsaro, D. C. Schmidt, R. Klefstad, and C. O'Ryan. Virtual Component: a Design Pattern for Memory-Constrained Embedded Applications. In *Proceedings of the* 9^{th} *Annual Conference on the Pattern Languages of Programs*, Monticello, Illinois, Sept. 2002.

[4] G. Daugherty. A proposal for the specialization of ha/dre systems. In *Proceedings of the ACM SIGPLAN 2004 Symposium on Partial Evaluation and Program Manipulation (PEPM 04)*, Verona, Italy, Aug. 2004. ACM.

[5] A. Memon, A. Porter, C. Yilmaz, A. Nagarajan, D. C. Schmidt, and B. Natarajan. Skoll: Distributed Continuous Quality Assurance. In *Proceedings of the 26th IEEE/ACM International Conference on Software Engineering*, Edinburgh, Scotland, May 2004. IEEE/ACM.

[6] Object Management Group. *Real-time CORBA Specification*, OMG Document formal/02-08-02 edition, Aug. 2002.

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