Supporting Configuration and Deployment of Component-based DRE Systems Using Frameworks, Models, and Aspects

Gan Deng
Department of EECS, Vanderbilt University
2015 Terrace Place, Nashville, TN, 37203 USA
01-615-343-7477
gan.deng@vanderbilt.edu

ABSTRACT

This research focuses on using frameworks, model-driven development, and aspect-oriented software development techniques to address key configuration and deployment concerns of component-based distributed real-time and embedded (DRE) systems. System designers and deployers can use these techniques to configure quality of service (QoS) aspects of their systems and fine-tune their systems during the design and runtime phases to ensure their systems meet end-to-end performance requirements.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Software Program Verification—*Design Tools and Techniques;* I.6.5 [**Simulation and Modeling**]: Model Development—M*odeling Methodologies*

General Terms: Design, Experimentation

Keywords: Component Middleware, Deployment and Configuration

1. DESCRIPTION OF PURPOSE

Component middleware helps enhance reuse by separating business logic concerns of components from their lifecycle management. Conventional component middleware, however, does not provide separation of QoS concerns (such as component server threading model, event dispatching priority model) from application business logic and lifecycle management, which is a requirement for distributed real-time and embedded (DRE) systems. Hence it is poorly suited for DRE systems where resources are constrained and variable over time. In particular, conventional component middleware lacks the capabilities to deploy and configure applications and middleware resources and services to provide the desired quality of service (QoS) to meet end-to-end system performance requirements. Our research addresses this problem by developing and integrating framework, model-driven development (MDD), and aspect-oriented software development (AOSD) techniques to separate QoS concerns and provide high level tools to weave these concerns into DRE systems and underlying middleware.

2. GOAL STATEMENT

The goal of this research is to develop and validate techniques that can deploy and configure DRE application components and component middleware to make component-based applications meet functional and QoS requirements more effectively than current techniques. In particular, this research explorers novel algorithms and technologies to (1) allocate the right execution environment resources so that the entire DRE system can meet end-to-end real-time QoS requirements, (2) configure, deploy, and effectively manage lifecycles of publish/subscribe services used by components at runtime to improve message exchange performance, and (3) support dynamic component assembly reconfiguration, redeployments, and migrations in the runtime without incurring noticeable performance degradation.

3. APPROACH

To achieve the goal described in section 2, DRE systems and middleware must be configured across multiple modularity boundaries, including multiple middleware layers. To simplify the problem we employ a layered architecture that combines middleware framework, models, and aspects. The lowest layer in this architecture is a reusable middleware framework called *adaptive configuration framework* (ACF) that provides highly-configurable middleware modules, which adaptively allocate system resources, and deploy and configure middleware services. The middle layer is an aspect-oriented *deployment and configuration engine* (DAnCE) that could weave various deployment and configuration concerns into the DRE systems and component middleware. The top layer is a domain-specific modeling language called the *service aspect modeling language* (SAML) that models system resource usages, publish/subscribe service configuration and deployment, as well as rules that determine how DRE systems migrate to adapt to changing environment. We describe the functionality of each layer and how they will collaborate with each other to effectively address various deployment and configuration concerns to make DRE systems meet end-to-end performance requirements.

Frameworks → Adaptive Configuration Framework

In our problem domain, to make the component middleware highly configurable and adaptable to different operating environments, we require *reflective techniques* to provide a greater degree of configurability and dynamic adaptability for the component middleware. The implementation of our novel reflective techniques is called the *adaptive configuration framework* (ACF), which (1) provides an integrated set of domain-specific structures and functionality based on patterns to improve system performance, (2) encapsulated and separates different deployment and configuration concerns of component-based applications so each concern could be manipulated independently without interfering with others.

The ACF consists of two reusable shared libraries, i.e., *resource allocation library,* and *service configuration and deployment* *library*. The resource allocation library configures component server resources by setting real-time policies defined in Real-time CORBA [3]. These server resources are specified via policies that designate application QoS requests including the number of containers [1] and their associated policies and the component server priority model, threading model, buffering model, and connection model. This module treats all these policies as first-class objects and allows them to be specified via input parameters of the exposed interfaces. By doing this, we can apply higher level system performance analysis and verification tools to ensure DRE systems are configured in an optimal manner.

The service configuration and deployment library configures various publish/subscribe services, such as those defined by OMG's Event Service, Real-time Notification Service, and Data Distribution Service (DDS), as well as the federated event services across multiple network domains. These services are used by many DRE systems and provide different levels of QoS guarantees. Our prior work [4] demonstrated that common middleware services can be configured using well-defined and documented CORBA interfaces and hence the usage patterns of such middleware services can be formulated. The ACF encapsulates such usage patterns and provides a reusable library that (1) contains a wrapper façade for the underlying publish/subscribe middleware services to shield component developers from tedious and error-prone programming tasks associated with initializing and configuring these publish/subscribe services, and (2) exposes interfaces to the external tools to manage the services so service configuration and deployment processes can be automated.

Aspects → An Aspect-oriented Deployment and Configuration Engine

The ACF provides a highly configurable and adaptable architecture and exposes certain interfaces to external tools that allow component server and middleware services to be configured. Since these configuration options tend to crosscut multiple modularity boundaries – including different layers of middleware and multiple stages of the DRE system lifecycle, such as compilation, deployment, and run-time (re)configuration – the use of aspects helps to detangle the configuration and customization logic of middleware from its functionality.

To help DRE system deployers to deploy and configure both the component middleware and component-based applications, we designed an aspect-oriented *deployment and configuration engine* (DAnCE), which is a meta-programmable-based approach that allows different deployment and configuration concern aspects, such as component server resource configuration, middleware service configuration, and component assembly reconfiguration, to be specified through metadata. DAnCE uses this metadata as input to a weaver that automatically inserts these crosscutting concerns into component middleware and component-based applications to drive the underlying service configuration framework.

To overcome the memory source constraints of most DRE systems, DAnCE can dynamically link or unlink necessary ACF libraries on demand at runtime by using the component configurator design pattern [5].

Models → Service Aspect Modeling Language

To simplify the development of component-based DRE systems, we are developing a high-level model-driven development (MDD) tool called *service aspect modeling language* (SAML), which supports the configuration, deployment, and validation of component middleware and applications. A key capability supported by SAML is the definition and implementation of a domain-specific modeling language (DSML), which uses concrete and abstract syntax to describe the concepts, relationships, and constraints used to express domain entities [3]. In particular, SAML allows system designers and deployers to (1) model DRE system resources, (2) specify real-time QoS policies and associate them with DRE systems, (3) specify publish/subscribe service behaviors and deployment requirements, and (4) model system migration rules so component assemblies and component middleware can be reconfigured and redeployment to adapt to changing environment. SAML enables visual manipulation of modeling elements and performs various types of generative actions, such as synthesizing XML-based deployment and configuration descriptors and synthesizing middleware service-specific configuration files.

4. EVALUATION

The contributions of this research include (1) developing an integrated approach that combines the three techniques described above to address various deployment and configuration concerns, (2) experimentally evaluate the advantages and disadvantages of the combined solution to other ad hoc solutions, and (3) discovery of various architectural and behavior patterns embodies in the integrated approach so other researchers in related research fields could evaluate and reuse. Finally, the results of this research are expected to introduce state-of-the-art design methodologies to the component based software engineering community. Our efforts are focused on the CORBA Component Model, particularly in the context of the *Component Integrated ACE ORB* (CIAO) [2] Lightweight CCM [1] implementation; however, the proposed approach can be extended to other component model as well.

5. REFERENCES

[1] Object Management Group: "Lightweight CORBA Component Model Revised Submission", *Object Management Group, Inc.* May 2003, realtime/03-05-05

[2] N. Wang, D. Schmidt, A. Gokhale, C. Gill, C. Rodrigues, B. Natarajan, J. Loyall, and R. Schantz, "QoS-enabled Middleware," *Middleware for Communications*, Wiley and Sons, New York.

[3] A. Ledeczi "The Generic Modeling Environment", *Workshop on Intelligent Signal Processing*, Budapest, Hungary, May 17, 2001.

[4] G. Edwards, G. Deng, D. Schmidt, A. Gokhale, and B. Natarajan, Model-driven Configuration and Deployment of Component Middleware Publisher/Subscriber Services, *Proceedings of the 3rd ACM International Conference on Generative Programming and Component Engineering*, Vancouver, CA, October 2004.

[5] D. 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.