Addressing Crosscutting Deployment and Configuration Concerns of Distributed Real-time and Embedded Systems via Aspect-oriented & Model-driven Software Development

Gan Deng  
Department of EECS  
Vanderbilt University  
2015 Terrace Place  
Nashville, TN 37203 USA  
gan.deng@Vanderbilt.Edu

Douglas C. Schmidt  
Department of EECS  
Vanderbilt University  
2015 Terrace Place  
Nashville, TN 37203 USA  
d.schmidt@Vanderbilt.Edu

Aniruddha Gokhale  
Department of EECS  
Vanderbilt University  
2015 Terrace Place  
Nashville, TN 37203 USA  
a.gokhale@Vanderbilt.Edu

ABSTRACT

Model-driven development (MDD) is gaining importance as an approach to resolving lifecycle challenges of large-scale distributed real-time and embedded (DRE) systems (e.g., avionics mission computing). DRE systems are characterized by their stringent requirements for quality of service (QoS), such as predictable end-to-end latencies, timeliness and scalability. Delivering the QoS needs of DRE systems entails the need to configure correctly, fine tune and provision the infrastructure used to host the DRE systems, which crosscuts different layers of middleware, operating systems and networks. Addressing these tangled deployment and configuration concerns of DRE systems requires integrating the principles of Aspect-Oriented Software Development (AOSD) with MDD. This demo showcases a set of software tools that resolve both the inherently and accidental complexities arising due to the configuration and deployment crosscutting concerns of component middleware-based DRE systems.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques: Model Driven; [Special-purpose and Application-based Systems]: Distributed Systems;

General Terms

Design, Performance

Keywords

Deployment and Configuration, Model-Driven Development, Component Middleware, Quality-of-Service

1. EMERGING TRENDS AND CHALLENGES

In recent years, quality of service (QoS)-enabled component middleware has emerged to help developers of DRE systems factor out reusable concerns to enhance reuse, such as component lifecycle management, authentication/authorization, and remoting. QoS-enabled component middleware technologies, such as Lightweight CCM [1], further simplify QoS provisioning via metadata and tools that help to (1) automate DRE system development lifecycle phases, such as packaging and assembly, and (2) improve component reusability and transport protocols. As a result, software for large-scale DRE systems is increasingly being assembled from reusable modular components available from commercial-off-the-shelf (COTS) providers, rather than developed manually from scratch.

Although QoS-enabled component middleware technology provides many powerful capabilities, it also yields the following challenges for developers of DRE systems:

1. **Heterogeneity.** Large-scale DRE systems often run on a variety of computing platforms that are interconnected by different types of networking technologies with varying QoS properties. The efficiency and predictability of DRE systems built using different infrastructure components varies according to the type of computing platform and interconnection technology.

2. **Increased variability.** Functional variabilities include different business-logic implementations of the same interfaces, e.g., cranes, moving belts, and forklifts in our inventory tracking system case study may vary in their ability to transport certain types of goods, depending on their weight, size, hazard-level, and other properties. Non-functional variabilities include the configuration of middleware services (such as naming, notification, security, or load balancing), real-time QoS-related configuration policies (such as priority, concurrency and real-time scheduling), and configuration of middleware internals itself (such as which middleware features/modules should be enabled for a particular application in a particular environment).

3. **Challenging QoS constraints.** DRE software controllers are increasingly replacing mechanical and human control of critical systems. These controllers must simultaneously support many challenging QoS constraints, including (1) real-time requirements, such as low latency and bounded jitter, (2) availability requirements, such as fault propagation/recovery across distribution boundaries, (3) security requirements, such as appropriate authentication and authorization, and (4) physical requirements, such as limited weight, power consumption, and memory footprint. For example, a distributed patient monitoring system requires predictable, reliable, and secure monitoring of patient health data that can be distributed in a timely manner to healthcare providers.

4. **Increased Scale.** As DRE systems are joined together to form large-scale “systems of systems,” developers rarely have in-depth knowledge of the entire system or an integrated view of all the subsystems and libraries. This myopia can cause them to implement suboptimal solutions that duplicate code unnecessarily, complicate system evolution, and violate key architectural principles (e.g., using implementation-specific

Copyright is held by the author/owner(s).

ICSE’06, May 20-28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.
functionality instead of public interfaces or prematurely committing to non-portable deployment policies). More importantly, the scale and complexity of DRE systems makes it infeasible to deploy them in disconnected, standalone configurations.

5. Dynamic operating conditions. Operating conditions for large-scale DRE systems can change dynamically, resulting in the need for appropriate adaptation and resource management strategies for continued successful system operation. In civilian contexts, for instance, power outages underscore the need to detect failures in a timely manner and adapt in real-time to maintain mission-critical power grid operations. In military contexts, likewise, a mission mode change or loss of functionality due to an attack in combat operations requires adaptation and resource reallocation to continue with mission-critical capabilities.

These common challenges of large-scale DRE systems requires developers to integrate different platforms and tools that solve essentially the same types of problems – yet are often non-portable and non-interoperable – without overcomplicating their solutions and exceeding project time and effort constraints. Likewise, developers of reusable middleware must also address these challenges when refactoring common capabilities from applications into effective reusable technologies and providing a portable operating environment for application developers. To maximize software reuse and productivity, therefore, increased scale and variability must be addressed by combining technologies that support alternative configurations and implementations of functionality more effectively.

2. SOLUTION APPROACH

Model-driven Development (MDD) technologies, such as the Object Management Group (OMG)’s Model-driven Architecture (MDA) [2], have emerged to address the different lifecycle challenges of DRE systems outlined above, which includes satisfying both the functional and QoS needs of DRE systems during design, development, testing, maintenance, and evolution stages. MDD helps resolve key software development and validation challenges encountered by component middleware and DRE systems by combining (1) metamodeling, which defines type systems that precisely express key abstract syntax characteristics and static semantic constraints associated with particular application domains, such as software defined radios, avionics mission computing, and inventory tracking, (2) domain-specific modeling languages (DSMLs), which provide programming notations that are guided by and extend metamodels to formalize the process of specifying application logic and QoS-related requirements in a domain, and (3) model transformations and code generation that automate and ensure the consistency of software implementations with analysis information associated with functional and QoS requirements captured by structural and behavioral models.

Figure 1 illustrates how a MDD tool chain coordinates with design-time analysis tools and run-time infrastructure to address DRE system lifecycle challenges described below:

- **Specification and implementation**, which enables application functionality specification, partitioning, and implementation as components.
- **Packaging**, which allows bundling a suite of software binary modules and metadata representing application components.
- **Installation**, which involves populating a repository with the packages required by the application.
- **Configuration**, which allows configuration of the packages with the appropriate parameters to satisfy the functional and systemic requirements of application without constraining to any physical resources.
- **Planning**, which makes appropriate deployment decisions including identifying the entities, such as CPUs, of the target environment where the packages will be deployed.
- **Preparation**, which moves the binaries to the identified entities of the target environment.
- **Launching**, which triggers the installed binaries and bringing the application to a ready state.
which is a metamodeling environment that defines the modeling components visually. These interacting components are connected component compositions, and establish connections among CoSMIC allows modelers to define component interfaces and shown in Figure 2.

address the concerns at different stages of DRE systems lifecycle specific modeling languages (DSMLs) and generative tools to MDD toolsuite illustrated in Figure 2 is a collection of domain-paradigms for each stage of the CoSMIC tool chain. The CoSMIC developed using the Generic Modeling Environment (GME) [4],

As shown in Figure 1, various stages of the DRE systems lifecycle are tangled with different layers of the infrastructure that host the DRE systems. MDD tools that use AOSD techniques can be used in untangling the crosscutting concerns at each stage of the DRE lifecycle, which helps improve productivity and time-to-market. In the Section 3 of this demo proposal, we will showcase how the CoSMIC AOSD & MDD tool suite addresses DRE systems lifecycle tangled concerns, and how DAEnCE runtime infrastructure works together with CoSMIC seamlessly to actually deploy and configure a running distributed environment while simultaneously satisfying the QoS constraints.

3. DESIGN AND IMPLEMENTATION

3.1 The Design and Implementation of CoSMIC AOSD & MDD Tool suite

The deployment and configuration of DRE systems typically spans both the application components and the underlying middleware, which often involves manual modification to meta-data that is split across multiple XML descriptors [3]. A key evolution and change management problem exists because of the inter-dependencies and crosscutting between descriptors, and the fact that manual modification of large XML files is error-prone. The Component Synthesis with Model Integrated Computing (CoSMIC) tools are developed using the Generic Modeling Environment (GME) [4], which is a metamodeling environment that defines the modeling paradigms for each stage of the CoSMIC tool chain. The CoSMIC MDD toolsuite illustrated in Figure 2 is a collection of domain-specific modeling languages (DSMLs) and generative tools to address the concerns at different stages of DRE systems lifecycle shown in Figure 2.

CoSMIC allows modelers to define component interfaces and component compositions, and establish connections among components visually. These interacting components are connected together to form a valid component assembly. The semantic rules associated with component assemblies are enforced by constraints defined in CoSMIC’s metamodel and model interpreter. Its metamodel defines static semantic rules that determine valid connections between components. Its model interpreter ensure the dynamic semantics of models built using CoSMIC, which can range from performing analysis of models to synthesizing code for components and their metadata.

CoSMIC contains multiple model interpreters, each performing a particular function. The most commonly used interpreter for DRE systems is the packaging interpreter, which generates XML descriptors to address various concerns in the CCM D&C specification. These XML descriptors include (1) component interface descriptors, which capture information about component interfaces including component ports, (2) component implementation descriptors, which capture information about component implementations, such as the dependencies and the connections among components, (3) implementation artifact descriptors, which capture information about implementation artifacts including dependencies between such artifacts, (4) component package descriptors, which capture information about grouping of multiple implementations of the same component interface into component packages, (5) package configuration descriptors, which capture information about specific configurations of such component packages, and (6) component domain descriptors, which capture information about the target environment in which the component-based application will be deployed.

The CoSMIC tool suite uses GME to enforce “correct by construction” techniques, as opposed to the “construct by correction” techniques commonly used by post-construction tools, such as compilers, source-level debuggers, and script validators. CoSMIC ensures that the rules of construction – and the models constructed according to these rules – can evolve together over time. Each CoSMIC tool synthesizes metadata in XML for use in the underlying middleware. The CoSMIC tool suite currently uses a platform-specific model approach that integrates the modeling technology with QoS-enabled component middleware, such as CIAO (available from Vanderbilt University) [4]. CoSMIC provides the capability to inter work with third party model checking tools, such as Cadena [5].

3.2 The Design and Implementation of DAEnCE Tool suite

To deploy a DRE system based on particular operating settings, system deployers must perform certain tasks based on the deployment profile, including (1) preparation, which takes the pre-built software package and brings it into a component software repository under the deployer’s control, (2) installation, which downloads the software components to component server processes that run in each node in the target environment, (3) configuration, which customizes properties of components on each node based on metadata in the
configuration capability, we developed a run-time middleware to support effective, automated DRE system deployment and (4) launching, which initializes software components with necessary computing resources, and connects the ports of the components that are distributed throughout the target environment based on metadata in the deployment profile and executes the whole component assembly.

Conventional techniques for deploying large-scale component-based DRE systems can incur both inherent and accidental complexities. Common inherent complexities involve (1) ensuring component runtime libraries are compatible with the hosting nodes (e.g., OS compatibility) and (2) creating the correct number of processes to host components based on the specified deployment profile (e.g., some software components might be hosted in the same process to improve performance, whereas other components might be hosted in different processes to improve system fault tolerance and reliability). Common accidental complexities stem from using ad hoc techniques for moving component runtime binaries and other dependent runtime libraries to the corresponding nodes for deployment. To perform these changes manually is not only tedious and error-prone, but also makes the deployment effort hard to reuse, e.g., there is no easy way to migrate one component running from one node/process to another when a deployment profile changes.

To support effective, automated DRE system deployment and configuration capability, we developed a run-time middleware tool suite called the Deployment And Configuration Engine (DAnCE), whose structure is shown in the right part of the Figure 3. As shown in this figure, DRE system developers can model various deployment and configuration concerns via CoSMIC toolsuite, which automatically generates the corresponding deployment and configuration profile for the designated system. DAnCE then takes the generated profile, and automatically deploys the system into the QoS-enabled component middleware platform. DAnCE consists of implementations of a set of runtime middleware services that deal with the instantiation, installation, setting up connections, monitoring, and termination of components on the nodes of the target environment, and it uses AOSD techniques to separate various concerns.

DAnCE takes a platform-independent approach and works for different component middleware platforms. To implement the platform-independent middleware services, DAnCE consists of a set of standards-based runtime interfaces [6]. Some interfaces (such as ExecutionManager and DomainApplicationManager) run at the global domain level, whereas others (such as NodeManager and NodeApplicationManager) run on each node. These interfaces together manage the lifecycle of the DRE system deployment process to help configure component servers on the individual nodes, install components into containers, and set up connections among components that may be distributed across multiple nodes.

When DRE system deployers instruct a global interface to deploy an ITS assembly, they must give the XML-based deployment profile generated by the CoSMIC. The global interface then uses this profile as input to populate a global deployment plan that describes a mapping of a configured component assembly into a target domain. This plan includes information about nodes where components will be deployed, the mapping of component to nodes, information about connections among component instances, and information about process collocation strategies and attribute configurations of components. Depending on the total number of nodes needed for a particular deployment, global deployment interfaces then split the global plan into multiple local (node-level) deployment plans and passes them to each node-level interface.

CoSMIC and DAnCE tools are open-source tools and are available for download at http://www.dre.vanderbilt.edu.

4. REFERENCES