Model-driven Tools for Dependability Management in Component-based Distributed Systems

Sumant Tambe[†], Akshay Dabholkar[†], Jaiganesh Balasubramanian[†], Aniruddha Gokhale[†], Douglas C. Schmidt[†]
[†]Department of EECS, Vanderbilt University, Nashville, TN, USA

I. Introduction

Emerging trends and challenges. Component-based software engineering supported by middleware technologies, such as CORBA Component Model (CCM) and Enterprise Java Beans (EJB), has emerged as a preferred way of developing enterprise distributed real-time and embedded (DRE) systems, such as smart buildings, modern office enterprises, and inflight entertainment systems. These systems consist of applications whose dependability requirements, such as availability and security, must be satisfied simultaneously to ensure dependable operation [1], [2].

For correct dependable operation of enterprise DRE systems, however, multiple dependability attributes must often be simultaneously satisfied. There are inherent challenges in satisfying multiple dependability attributes together due to tradeoffs and conflicts between them. For example, deploying replicas of a service on hosts that are unauthorized to access by clients may result in unavailability of the service to its clients on failure of another replica of the same service. It is hard to detect and analyze these errors at runtime, which motivates the need to catch as many errors at design-time as possible. Hence, there is a need for design-time tools to reason about the inherent tradeoffs and conflicts between multiple dependability attributes and alleviate complexities in developing dependable enterprise DRE systems.

Figure 1 depicts desirable properties of a design-time tool that considers multiple dependability attributes (such as availability and security) to reason about system dependability *e.g.*, protecting it from various hazards (such as faults and unauthorized access).

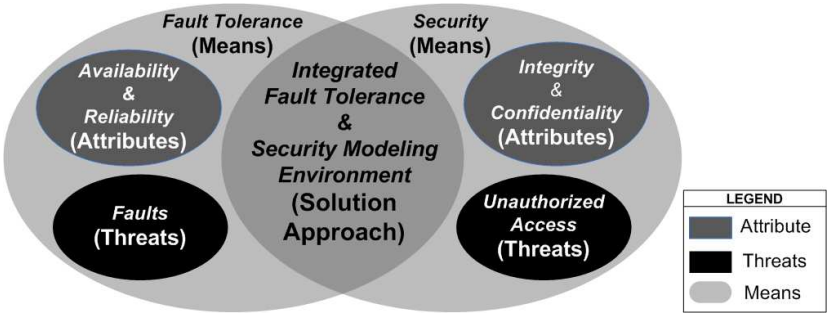

Fig. 1: **Dependability Solution Space**

In addition to addressing these challenges, there are also challenges associated with managing metadata that compose, deploy and configure the systems in accordance with design-time tradeoffs on the underlying middleware that provides a component-based programming model. Developers must supply correct metadata to ensure that the system satisfies its dependability requirements. These challenges are exacerbated by component middleware that provides multiple levels of granularity (such as the component-level, port-level, assembly-level and connection-level) at which dependability attributes can apply.

Solution approach → Unified dependability modeling and reasoning using model-driven engineering. This chapter describes how model-driven techniques and tools can provide effective mechanisms to handle multiple dependability attributes all at once and make design-time tradeoffs between the conflicting requirements of these attributes. It also demonstrates how different levels of granularity offered by the component middleware are handled simultaneously by the model-driven tools.

To make the discussions concrete, we showcase model-driven dependability techniques in the context of a tool we developed called *Model-based Provisioning Engine for Dependability* (MoPED). MoPED provides intuitive abstractions that help software developers and systems engineers model and reason about the availability and security requirements simultaneously. MoPED provides a domain-specific modeling language (DSML) to represent the key architectural abstractions of component-based systems and their dependability requirements expressed as availability and security attributes. MoPED also bridges the gap between high-level system requirements and configuration of low-level middleware mechanisms.

II. Outline of the Remainder of the Chapter

Section 2: Satisfying Dependability Requirements of DRE Systems – A Case Study Perspective. This section describes key challenges in developing dependable enterprise DRE systems by presenting a case study of a representative enterprise security

and hazard sensing system. The challenges associated with deploying and configuring such a system with integrated security *and* availability requirements are demonstrated using concrete scenarios within the enterprise security and hazard sensing system. Based on these scenarios and challenges, this section summarizes and highlights the problems of manually transforming the described high-level dependability requirements of the scenario into declarative metadata that configures low-level component middleware mechanisms. The solutions provided by MoPED to solve those problems are discussed in subsequent sections of the chapter.

Section 3: DSML Support for Dependability. Non-functional DRE system requirements, such as availability and security, must be satisfied to ensure their dependable operation. For example, caller access rights must be verified before invoking a method on a component protected by access control policies. These non-functional requirements manifest themselves at several levels of granularity ranging from organizational domain to individual component methods. For example, access control policies could be applied on methods, ports, components, and component assemblies. Similarly, availability requirements are typically applied at component or assembly level in terms of the number of replicas desired.

Supporting multiple dependability attributes requires the DSML to support some key properties. First, it should provide built-in constraints to handle several key challenges (such as adding constraints to allow composition of multiple dependability attributes, *e.g.*, failure recovery at the component level and security at the interface level) that must be addressed when combining multiple attributes of dependability. Second, it should provide generative capabilities so that platform-specific metadata for deployment and configuration that accurately reflects the designer's choice of dependability requirements can be automatically synthesized.

A promising solution is to have the DSML provide first class support for rich component-based domain-specific abstractions so that it can help capture the requirements of component-based dependable systems. These abstractions should be able to account for the diversity in component platforms. When the choice of implementation technology is made, dependability requirements should be transformed into mechanisms and policies of the selected component technology.

To provide a concrete exemplar of DSML support for dependability, this section describes the design of *Model-based Provisioning Engine for Dependability* (MoPED), which is an extensible modeling framework that allows component-based system developers to express dependability design intent at different levels of granularity using intuitive visual representations. MoPED is developed using the Generic Modeling Environment (GME) [3], which is a meta-programmable tool for developing DSMLs. This section describes MoPED's three key capabilities: (1) domain-specific, QoS modeling support for component-based systems, (2) unified availability and security modeling and reasoning support, and (3) extensible modeling language and tool support.

MoPED provides a DSML that is based on mandatory and optional features present across contemporary component-based middleware infrastructures. Contemporary component infrastructures, such as EJB, CCM support all the mandatory features: components, connections, remotely invocable methods, and a notion of deployment. Moreover, CCM supports the optional features (*e.g.*, port and assembly) as well. MoPED can be used to model dependability requirements for target component infrastructures that support all the mandatory features, and optionally ports and assembly.

In MoPED's model-driven software development process, the modeling language, its tool support, and instance models are the primary software artifacts that developers manipulate at design-time. Similar to the object-oriented paradigm—where modularized and extensible design is integral for system maintainability—MoPED's DSML and tool support are extensible to accommodate new dependability requirements that arise as systems evolve.

By using MoPED, developers can defer the choice of implementation technology to later stages of software development, which simplifies (1) application code by decoupling dependability aspects from application functionality and (2) application deployment and configuration by conducting dependability analysis irrespective of the deployment platform.

Figure 2 shows a simplified[1] metamodel of MoPED's DSML. This metamodel shows ≪Model≫ stereotypes of component, assembly, port, and method, which serve as placeholders for components, assembly, ports, and method abstractions, respectively, provided by the underlying middleware technology.

A classification of QoS models that can be associated with basic component middleware abstractions is made using a set of abstract QoS elements denoted using the ≪FCO≫ (First Class Object) stereotype in Figure 2. The abstract QoS elements in the metamodel (e.g., *ComponentQoS*, *AssemblyQoS*, *PortQoS*, *MethodQoS*) are used as base classes for concrete QoS models.

Based on the extensible QoS modeling framework, MoPED's DSML provides concrete QoS models that capture availability and security requirements of a component-based system at different levels of granularity, such as components, connections, methods and optionally ports and assemblies. Constraints written in the Object Constraint Language (OCL) help designers avoid modeling conflicting availability and security design decisions.

Section 4: Dependability Reasoning Using MoPED. This section describes how MoPED provides capabilities to analyze the tradeoffs between multiple dependability attributes of enterprise DRE systems, and eases the development of such systems. MoPED's model-driven design and development process consists of the following steps, which are illustrated in the context of the enterprise security and hazard sensing system:

1. Modeling availability requirements. Unlike the traditional client/server model of designing distributed systems, component-

[1]Some elements and associations present in the original metamodel are removed due to space considerations.

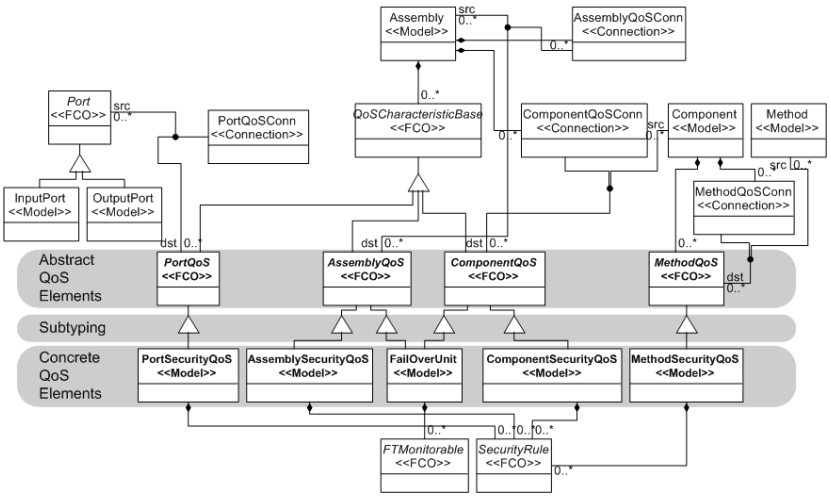

Fig. 2: **Simplified Metamodel of MoPED's DSML**

based systems often have more than one component arranged in a workflow-like pattern (assembly) to realize critical application functionality. In the event of a failure of one component, the entire assembly needs to fail over to a replica assembly instead of a replica of the failed component to avoid cascaded failures. The granularity of protection for component-based systems is an assembly, which could be part of a single process or spread across multiple processes on multiple machines.

MoPED provides modeling capabilities to group set of components into a *FailOverUnit* to specify the granularity of fault-tolerance in enterprise DRE systems. Moreover, a *FailOverUnit* does not require modeling of replicas; only the desired number of replicas (*i.e.*, the replication degree) need to be provided. Depending upon the replication degree, MoPED tool chain generates the necessary number of replicas of the assembly and all its constituent components automatically. While doing so, it also automatically generates complex connection topology interconnecting the generated components, which is dictated by the replication degree of the primary component and replication degree of components that it interacts with.

2. Modeling security requirements. Since security is an essential aspect of a dependable component-based system, it must be configured and enforced at different levels of the system granularity, such as organizational domain, its subdomains, application assemblies, components, and remotely invocable methods in components.

Support for security QoS modeling in MoPED focuses on two attributes of security: confidentiality and integrity. MoPED provides a *role-based access policy* (RBAP) model to define *role-based access control* (RBAC) for enterprise systems and to provide secure transport protocol configurations for data integrity. MoPED's RBAP model is motivated and designed in response to the OMG's RBAP Metamodel RFP [4].

3. Integrated reasoning of availability and security requirements. The key contribution of MoPED's dependability QoS support is the unification of the RBAP model and secure transport protocol configurations with availability requirements modeling. Security QoS leverages MoPED's constraint-checking mechanisms to detect design-time errors in security configurations. MoPED also validates the decisions taken by security modeling against the decisions taken by availability modeling. The inherent challenge in integrating availability and security stems from the fact that they are often tangled with each other and higher level analysis is necessary to resolve the conflicts between them at design-time. MoPED's design environment uses constraints written using OCL to check every design decision taken by the QoS modelers. MoPED checks the availability and security QoS requirements in the model against OCL constraints to detect possible conflicts.

Section 5: Related Work. This section provides a survey of related work that focuses on model-based provisioning of computer system dependability. Related research within the areas of dependability modeling and analysis, and model-driven dependability provisioning techniques will be compared and reviewed with the context of our MoPED approach. For example, MEAD [5] and AQUA [6] provide run-time solutions for dynamic adaptation of fault-tolerance properties in response to changing resource availabilities. Likewise, The OMG's Model-driven Architecture (MDA) and Unified Modeling Language (UML) profiles can provide design-time solutions to model either (1) availability requirements [7], [8] or (2) security requirements [9], [10] to perform predictive analysis of a system's dependability properties.

Section 6: Concluding Remarks. will present concluding remarks that summarizes how model-driven tools and techniques can provide effective mechanisms to make design-time tradeoffs among the conflicting requirements of multiple dependability attributes. It will also present lessons learned from our experience developing MoPED. Finally, it will describe potential extensions to MoPED to address some of its limitations. For example, model-to-model transformation approaches to analyze high-level requirements using formal analysis tools could be explored to quantitatively evaluate the dependable system design and developed using MoPED. We are exploring ways to integrate multi-dimensional tradeoff analysis for different dependability attributes in MoPED.

REFERENCES

[1] A. Avizienis, J. Laprie, and B. Randell, "Fundamental concepts of dependability," 2001. [Online]. Available: citeseer.ist.psu.edu/article/avizienis01fundamental.html

[2] B. Littlewood and L. Strigini, "Software reliability and dependability: a roadmap," in *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA: ACM, 2000, pp. 175–188.

[3] Á. Lédeczi, Á. Bakay, M. Maróti, P. Völgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing Domain-Specific Design Environments," *Computer*, vol. 34, no. 11, pp. 44–51, 2001.

[4] Object Management Group, "Role Based Access Policy (RBAP) Metamodel RFP," http://www.omg.org/cgi-bin/doc?bmi/2008-02-07, 2008.

[5] P. Narasimhan, T. Dumitras, A. Paulos, S. Pertet, C. Reverte, J. Slember, and D. Srivastava, "MEAD: Support for Real-time Fault-Tolerant CORBA," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 12, pp. 1527–1545, 2005.

[6] Y. Ren, D. Bakken, T. Courtney, M. Cukier, D. Karr, P. Rubel, C. Sabnis, W. Sanders, R. Schantz, and M. Seri, "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects," *Computers, IEEE Transactions on*, vol. 52, no. 1, pp. 31–50, 2003.

[7] A. Bondavalli, I. Mura, and I. Majzik, "Automatic dependability analysis for supporting design decisions in uml," in *HASE '99: The 4th IEEE International Symposium on High-Assurance Systems Engineering*. Washington, DC, USA: IEEE Computer Society, 1999, p. 64.

[8] G.Rodrigues, "A Model Driven Approach for Software Systems Reliability," in *In the proceedings of the 26th ICSE/Doctoral Symposium, May 2004 - Edinburgh, Scotland*. ACM Press, May 2004.

[9] J. Jürjens, "Umlsec: Extending uml for secure systems development," in *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*. London, UK: Springer-Verlag, 2002, pp. 412–425.

[10] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From uml models to access control infrastructures," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 1, pp. 39–91, 2006.

III. RELEVANCE FOR THIS BOOK

This proposed chapter describes a practical approach to modeling key attributes of dependable systems (availability and security) via a unified QoS modeling framework. This approach is presented in the context of a model-driven tool chain called MoPED that provides intuitive, domain-specific modeling abstractions to capture availability, security and network level QoS requirements of component-based systems. The techniques used in MoPED are within the context of model-driven engineering, such as domain-specific modeling, and widely used component middleware, such as CCM, EJB, and J2EE. This chapter can serve as a novel demonstration of using model-driven engineering to address dependability issues in component-based enterprise DRE systems, despite the variabilities in the component platforms used to develop such systems.

IV. AUTHOR BIOGRAPHIES

Sumant Tambe is a Ph.D. student at the department of EECS at Vanderbilt University. He received his MS (Computer Science) from New Mexico State University, Las Cruces, NM. His research interests are Model-driven Engineering (MDE), QoS requirements modeling and provisioning for DRE systems, and model-driven development and deployment of fault-tolerant distributed systems.

Akshay Dabholkar is a Ph.D. student at the department of EECS at Vanderbilt University where he received his MS (Computer Science). His research interests are Feature-oriented Middleware Specialization using Model-driven Engineering techniques for varied domains, QoS requirements modeling and provisioning for DRE systems.

Jaiganesh Balasubramanian is a Ph.D. student at the department of EECS at Vanderbilt University. He received his MS (Computer Science) from University of California, Irvine. His research interests are in designing, and developing, algorithms and middleware architectures for adaptive resource management in real-time fault-tolerant systems, and model-driven deployment and configuration of distributed systems.

Dr. Aniruddha S. Gokhale is an Assistant Professor of Computer Science and Engineering in the Department of Electrical Engineering and Computer Science at Vanderbilt University. He received his BE (Computer Eng) from Pune University; MS (Computer Science) from Arizona State University; and D.Sc (Computer Science) from Washington University. Prior to joining Vanderbilt, he was a Member of Technical Staff at Bell Labs, Lucent Technologies. Dr. Gokhale is a member of IEEE and ACM. Dr. Gokhale's research combines model-driven engineering and middleware for distributed systems, notably real-time and embedded systems. He is the project leader for the CoSMIC model-driven engineering tool suite at Vanderbilt.

Dr. Douglas C. Schmidt is a Professor of Computer Science and Associate Chair of the Computer Science and Engineering program at Vanderbilt University. He has published 9 books and over 400 papers that cover a range of topics, including patterns, optimization techniques, and empirical analyses of software frameworks and domain-specific modeling environments that facilitate the development of distributed real-time and embedded (DRE) middleware and applications. Dr. Schmidt has over 15 years of experience leading the development of ACE, TAO, CIAO, and CoSMIC, which are open- source middleware frameworks and model-driven tools that implement patterns and product-line architectures for high- performance DRE systems.