MoPED: A Model-based Provisioning Engine for Dependability in Component-based Distributed Real-time Embedded Systems

Sumant Tambe
Dept. of EECS
Vanderbilt University
Nashville, TN, USA
sutambe@dre.vanderbilt.edu

Akshay Dabholkar
Dept. of EECS
Vanderbilt University
Nashville, TN, USA
aky@dre.vanderbilt.edu

Aniruddha Gokhale
Dept. of EECS
Vanderbilt University
Nashville, TN, USA
gokhale@dre.vanderbilt.edu

***Abstract*—Developing dependable distributed real-time and embedded (DRE) systems incurs significant complexities in the tradeoffs resulting from the different conflicting attributes of dependability, such as predictability, availability, and security. In component-based systems, these challenges are exacerbated since the tradeoffs must faithfully be reflected within the complex metadata descriptors used to compose, deploy and configure the system. The benefits of design-time approaches to address these problems are well-understood. Existing model-driven design-time tools for developing dependable systems, however, focus largely on only one dependability attribute at a time and lack of extensibility results in rigid and hard to maintain tool support.**

This paper describes MoPED (Model-based Provisioning Engine for Dependability), which is a model-driven framework that unifies reasoning about predictability, availability, and security requirements for developing dependable component-based DRE systems. We evaluate the capabilities of MoPED using a representative case study and show how it alleviates complexities in the design of dependable systems and reduces manual efforts in the deployment phase by an order of magnitude.

I. Introduction

Emerging trends and challenges. Component-based software engineering supported by middleware technologies (*e.g.,* CORBA Component Model (CCM)) have emerged as a preferred way of developing distributed real-time and embedded (DRE) systems, such as shipboard computing systems, enterprise security and hazard sensing systems, and intrusion-tolerance systems. These systems consist of applications whose quality of service (QoS) requirements – notably predictability, availability, and security, must be satisfied simultaneously to ensure dependable operation [2], [14].

Prior research has focused on design- and run-time solutions to address the problem of assuring dependability of distributed systems. For example, OMG's Model-driven Architecture (MDA) and UML profiles can provide design-time solutions to model either (1) predictability requirements [19], (2) availability requirements [5], [8] and (3) security requirements [4], [11] to perform predictive analysis of a system's dependability properties. Likewise, MEAD [15] and ARMOR [12] provide run-time solutions for dynamic adaptation of fault-tolerance properties in response to changing resource availabilities.

For correct dependable operation of 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 the primary 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. The design-time tools must be able to reason about the inherent tradeoffs and conflicts between multiple dependability attributes.

In addition to addressing such inherent challenges, accidental complexities are incurred due to complexity of managing metadata that compose, deploy and configure the systems on the underlying component-based middleware in accordance with design-time tradeoffs. Developers must supply correct metadata to ensure that the system satisfies its dependability requirements. These accidental complexities are exacerbated since component-based middleware 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. To address the challenges above, we have developed a model-driven engineering (MDE) design tool called *Model-based Provisioning Engine for Dependability* (MoPED) that provides high level, intuitive abstractions to model and reason about the availability and security requirements while maintaining predictability. MoPED provides a domain-specific modeling language (DSML) for modeling the key architectural abstractions of component-based systems and their dependability requirements expressed as availability and security attributes. Moreover, it bridges the gap between high-level system requirements and configuration of low-level middleware mechanisms.

Paper organization. The remainder of the paper is organized as follows: Section II motivates the need for a design-

time modeling and reasoning tool such as MoPED via a detailed case study; Section III describes the challenges we faced in the case study and how we address them using MoPED; Section V quantitatively evaluates MoPED in the context of the case study; Section VI compares MoPED with related work; and Section VII presents concluding remarks.

II. Satisfying Dependability Requirements of DRE Systems: A Case Study Perspective

Figure 1 shows a representative DRE system in an office enterprise security and hazard sensing environment. This section

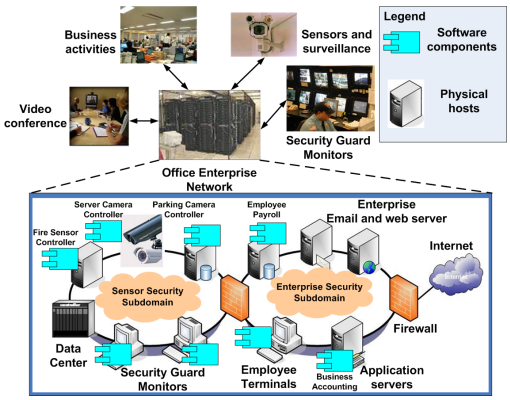

Fig. 1. An Enterprise Security and Hazard Sensing System

describes key inherent and accidental challenges that must be met at design-time when addressing dependability criteria for such DRE systems.

(1) System deployment and configuration. Deploying and configuring office enterprise security and hazard sensing systems involves addressing the following requirements of its constituent subsystems:

a. Business application subsystem. Figure 1 shows *Business Accounting* and *Employee Payroll* application components, enterprise servers (email and web servers), and employee terminal components that must be deployed in the *Business* security subdomain.

b. Security surveillance and hazard sensing subsystem. Figure 1 shows the *server*, *parking camera controllers* and user interface components of the security guard monitors. The software components belonging to this subsystem must be deployed in the *Sensor* security subdomain, which has stricter access control policies than *Business* security subdomain where normal business operations run. Moreover, the criticality of this subsystem requires it to be highly available and predictable. Individual tasks in the sensor application, such as the fire sensor task, sensor controller task, and the display task have end-to-end soft real-time deadlines, which must be met in all but exceptional (failure) scenarios.

(2) Security and availability tradeoffs. The system has multiple users organized into a hierarchy of roles, such as administrator, developer, manager, and regular employees. Although *Business* and *Sensor* security subdomains separate the two subsystems, components in one security subdomain often require access to components deployed in the other security subdomain. For example, access control policies should allow access to the *parking camera controller* from employee terminals but prohibit access to the *server camera controller*. Similarly, network administrators may have access to the *server camera controller* but not to the *fire sensor controller*, which is accessible only to the security guard monitors.

Along with the security access control requirements, availability requirements of the enterprise office system requires replication of critical software components to improve availability. For example, the *Business Accounting* and *Employee Payroll* components should be replicated. To avoid violating security requirements, however, these components must be deployed on hosts belonging to the same security subdomain.

In the various use cases above, it is tedious and error-prone to transform the high-level dependability requirements of the scenario described above into declarative metadata that configures low-level component middleware mechanisms. This process is even more complicated when security and availability attributes must be handled simultaneously. In the following section, we show how MoPED resolves these challenges.

III. Model-based Provisioning for Dependability

This section describes the design of *Model-based Provisioning Engine for Dependability* (MoPED), which is a 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) [13], which is a meta-programmable tool for developing DSMLs. MoPED provides three main capabilities: (1) domain-specific, QoS modeling support, (2) unified availability and security modeling support, and (3) ensuring predictability of the real-time components using schedulability analysis.

.

A. DSML Support for Dependability Using MoPED

We now identify the need for a domain-specific approach for modeling QoS attributes of dependability. We describe our solution approach based on a feature model [7] of component-based systems.

1) Challenge: Dependability modeling support for component-based systems: 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, such as methods, ports (interfaces), components, and component assemblies. Similarly, availability requirements are typically applied at component or assembly level in terms of the number of replicas desired.

A domain-specific approach can help capture the requirements of dependable systems. This approach should leverage rich component-based abstractions and composition mechanisms provided by nearly all component platforms. When the

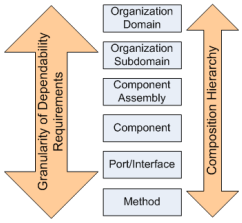

Fig. 2. **Different Levels of Granularity for Dependability Specification.**

choice of implementation technology is made, dependability requirements should be transformed into mechanisms and policies of the selected component technology.

2) Resolution: A component-based approach for modeling dependability requirements.: MoPED leverages the Component Quality of Service Modeling Language (CQML) [25] DSML that provides a feature model based on mandatory and optional features present across contemporary component-based middleware infrastructures. Contemporary component infrastructures, such as EJB, CCM support all the mandatory CQML features such as components, connections, remotely invocable methods, and a notion of deployment. Moreover, CCM supports the optional CQML features (*e.g.*, port and assembly) as well. Hence, MoPED can be used to model dependability requirements for target component infrastructures that support all the mandatory features, and optionally ports and assembly. The abstract CQML QoS elements form the core of the unified dependability modeling capabilities of MoPED's DSML. Moreover, these abstract QoS elements serve as the points of extension, expressed using subtype relationship in the QoS modeling framework.

Concrete models for availability and security attributes are defined in CQML as extensions of the QoS modeling framework. Availability requirements, such as number of replicas, are captured using a *FailOverUnit*, whereas security access control requirements are captured using *PortSecurityQoS*, *AssemblySecurityQoS*, *ComponentSecurityQoS*, and *MethodSecurityQoS*. Section III-B describes the availability and security semantics that these concrete models capture. The subtype relationship allows the FailOverUnit and other security models to reuse generic syntactic and semantic constraints defined on their parent abstract QoS types, giving rise to a consistent and unified dependability modeling environment.

B. Modeling Security and Availability Requirements Simultaneously

Experience has shown that integrating security features into a fault-tolerant system and vice versa should not be an after thought. It is much more complicated than pre-planning desired dependability and security features during a project's specification phase.

1) Challenge: Lack of unified modeling support.: Dependability comprises multiple attributes, such as availability and security, that must all be assured simultaneously for proper system operation. Modeling individual dependability attributes independently and reasoning about them in isolation can yield systems that do not meet their QoS objectives when deployed. It is therefore important model and reason about dependability capabilities simultaneously.

A challenge in developing integrated modeling support stems from the fact that dependability measures are often tangled with each other and cross-cut with the primary functional dimension of system decomposition. Formal approaches addressing this challenge have limited success in protocol analysis [22] and dependability evaluation [16] due to the difficulty in using formal approaches without strong tool support. Below we describe how MoPED simplifies unified dependability modeling by means of intuitive abstractions and tool supported guidance to modelers.

2) Resolution: Unified dependability modeling using MoPED.: Based on the QoS modeling framework of CQML, 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.

(1) Modeling availability requirements. Unlike the traditional client/server model of designing distributed systems, component-based systems often have more than one component arranged in a workflow-like pattern (assembly) to realize critical application functionality. To simultaneously meet the predictability and availability requirements of the end-to-end application workflows, group-failover [23] protocol has been developed. In the event of a failure, group-failover protocol allows the clients to failover to a replica assembly to maintain the state consistency and timeliness of application data. Therefore, the granularity of protection for component-based systems is a group of components (assembly), which could be part of a single process or spread across multiple processes on multiple machines.

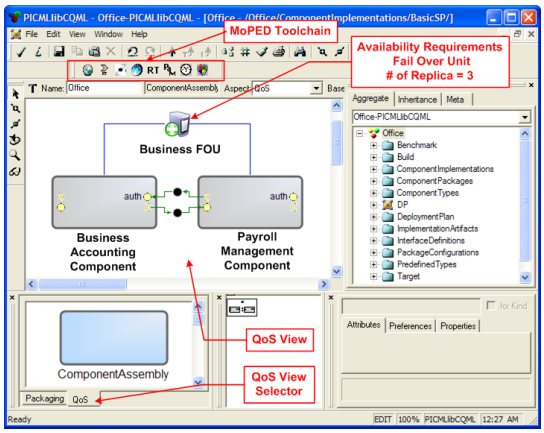

Fig. 3. **Availability Requirements Modeling Using MoPED**

A *FailOverUnit* is the key availability modeling abstraction supported in MoPED that controls the granularity of protection. One or more components and assemblies can be

associated with a *FailOverUnit*. For example, as shown in Figure 1, our case study has two critical components of an intranet application namely, *Employee Payroll* and *Business Accounting*. If one component fails, the assembly needs to fail over to its replica.

In the MoPED's design environment, the modeling of *FailOverUnit* is done in the *QoS* view, which avoid tangling of availability concerns with the composition concerns at modeling time. Moreover, a *FailOverUnit* does not require modeling of replicas; only the desired number of replicas (*i.e.*, the replication degree) need 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. A detailed discussion of the model transformation algorithms that generate the replica component topologies and complex interconnections between them are described in [24], [26].

(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 inspired from the OMG's RBAP Metamodel RFP [21]. Security domains and sub-domains are also supported to simplify management of security policies spanning enterprise-wide systems, such as our case study.

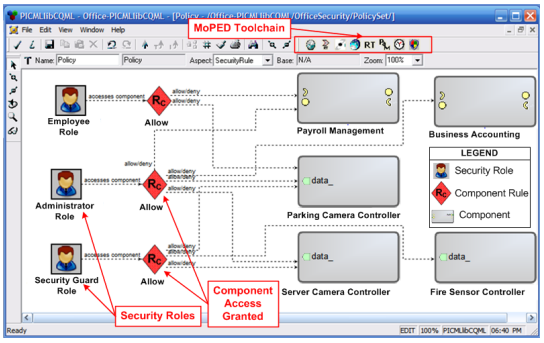

Fig. 4. **Security Access Control Modeling Using MoPED**

MoPED's RBAP model provides a central place to model access control policies for an enterprise. It specifies associations between users and their roles, roles and their domain level rights, and resources in a security domain along with the required permissions to access them. It is used to define the rules that dictate which roles (possessing distinct rights) are able to access the different resources, such as application assemblies, components, and ports within the system.

MoPED's security QoS modeling supports fine granularity of component-based systems as it is built as an extension to MoPED's core QoS modeling framework. *ComponentSecurityQoS* is used to model RBAP over the components and attributes. Similarly, *AssemblySecurityQoS* is used to model RBAP over an entire application assembly or subassembly. It ensures consistent security policies across entire workflow of constituent components as an end-to-end measure. *ConnectionSecurityQoS*, conversely, models secure transport protocol configuration for inter-component interactions.

Security QoS abstractions associate required access rights to the various key abstractions (*e.g.*, components, ports) of the component-based system, as well as define rules that control the access rights of the various roles to these system elements. For example, the *Administrator* role has the responsibility of managing the office business process application servers that consist of the *Business Accounting* and *Employee Payroll* components. These rules are encapsulated as the system security policies shown in the Figure 4. MoPED automatically generates correct deployment metadata related to defined mappings, permissions and security policies using existing OASIS [17] standardized formats for supporting various runtime mechanisms for implementation of RBAC.

(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. The possible conflicts are of two types: violation of security policies due to an availability decision or vice versa. Their bidirectional interdependencies are shown in the following two examples:

- **Detecting security QoS violations.** MoPED prevents designers from placing any replica of a component in a different security subdomain than that of the primary component because access policies across subdomains are usually quite different. Deploying replicas in a different subdomain could lead to unavailability of a service upon failure due to difference in the security privileges across two different subdomains. For example, the replicas of the *Business Accounting* component cannot be deployed

Listing 1 Using OCL to Detect Conflicts in Dependability Modeling

```
-- OnError: "Deployment hosts of replicas must belong to the same security subdomain."
context c: Component        -- For each component in the model.
let fou = c.connectedFCOs(FailOverUnit) in -- Find FailOverUnit associated with c, if any.
let replicas = getReplicaSet(c,fou.replicaCount()) in -- Get set of replicas and primary
    replicas->forAll(i,j : Component | -- For each pair of the replicas
    let nodeA = i.connectedFCOs(Host) in -- Find deployment host for i
    let nodeB = j.connectedFCOs(Host) in -- Find deployment host for j nodeA.SecurityDomain.name().trim()
              = nodeB.SecurityDomain.name().trim()) -- Name of security domain must be the same
```

in the *Sensors* security subdomain, which has much more restricted access compared to the *Business* security subdomain.

The MoPED design environment detects such security violations using OCL constraints and provides guidance in the form of an error message to fix them. An example OCL constraint with comments and the guidance message is shown in Listing 1. Similarly, MoPED verifies that components that communicate across security subdomains have the necessary access rights to do so.

- **Consistent duplication of access control decisions.** The access control security policies defined for a primary component and secure transport configuration for connections should be consistently replicated for their replicas too. MoPED analyzes security QoS attributes and automatically duplicates the associated policies. Without an automated support for consistent security policy duplication, it is tedious and error-prone for security modelers to ensure that all replicas have consistent access control decisions.

The interdependencies and conflicts between availability and security resolved at design-time prevents dependability issues in the later stages of system lifecycle, such as testing and production.

IV. Analyzing Dependability Trade-offs for Maintaining Predictability

In this section we describe how MoPED can be used to analyze the schedulability of the system models and how availability and security trade-offs can be made. We use the scenario described in Section II to illustrate how MoPED reasons about timeliness and generates metadata that are used by a back-end real-time schedulability analysis tool. We chose the real-time QoS aspect as the primary aspect and compute the effects of dependability requirements on it for analysis. It allows temporal correctness to be integrated during development, rather than the more typical practice to testing timeliness at the end of development. It avoids the costly problems that can arise when timing faults are found later during testing or, still worst, after deployment.

We have currently integrated with the Times [1] schedulability and model checking functionality through the RTAnalysis interpreter. In order to correctly determine the schedulability of the system it is important to convey the behavioral semantics of the system components. We leverage the behavior modeling capability provided by an input/output automata-based language called Component Behavioral Modeling Language [9]. We use it to produce high-fidelity mapping of component behavior into timed automata–the underlying formalism used by the Times tool. Alternatively, the behavior could be fed into the interpreter through other formalisms, such as UML state charts and/or activity diagrams.

The RTAnalysis interpreter obtains the data necessary for schedulability analysis from both: CQML's *RealTimeConfiguration* models and the behavior model of components. It requires the task priorities, execution times, task behavior types (periodic, sporadic or controlled), and deadline to generate input for the Times tool, from which the tool derives worst-case response times (WCRTs). We depend on the modeler to provide the above mentioned information.

Schedulability under Dependability Requirements. We now describe how we performed schedulability analysis of the Enterprise case study considering both the availability and security requirements. Quite often, the DRE systems that have simultaneous dependability requirements, trade offs have to be made in the quality levels of different QoS aspects of the system.

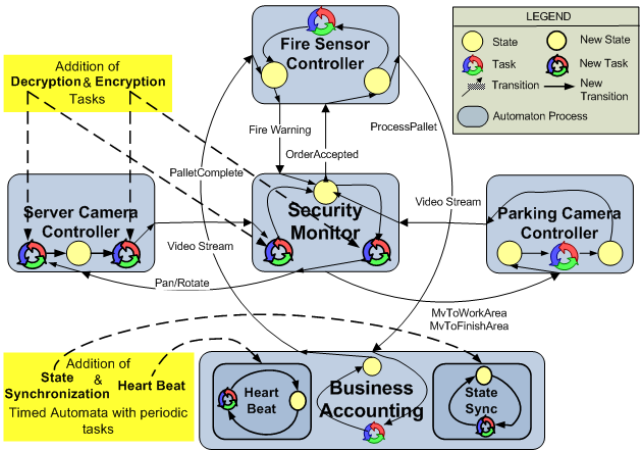

Fig. 5. **Effects of Multi-QoS Interweaving in Enterprise Case Study**

In the dependability models of MoPED, the frequency of heart-beat beacon determines the quality of the fault-monitoring infrastructure with respect to the fault detection rate. In order to improve the fault detection rate, an increase in the heart-beat frequency, might have an adverse effect on the schedulability of the critical path. The reason being, with the increase in the fault-monitoring frequency, there is a corresponding increase in the number of instances of the corresponding periodic tasks to be scheduled within the deadline.

The RTAnalysis interpreter responds to the *periodic computation* events and produces an updated timed automata of the system behavior as shown in Figure 5. The behavioral semantics map to new process automata containing new tasks and states for the `Business Accounting` component. Likewise, while weaving the security QoS aspect in `Security Monitor` and `Server Camera Controller` components, the additional CPU overhead of encryption/decryption must be incorporated at the port level boundaries of above components. The RTAnalysis interpreter adds an encryption task right before leaving the `Security Monitor` automaton and a decryption task right after entering the `Server Camera Controller` automaton.

This new model can be analyzed by the Times tool to determine whether the system is still schedulable *i.e.,* meets its real-time deadlines. Thus the interpreter automates the process of determining the effect on system schedulability due to interweaving of other QoS aspects such as fault-tolerance and security. The results from the analysis can be used to fine tune the system's QoS aspect configuration, for example, the heart-beat beacon frequency and/or key length.

V. Evaluating MoPED

This section describes our evaluation of MoPED. We show how modeling dependability requirements using MoPED alleviates the tedious and error-prone effort of manually writing platform specific metadata.

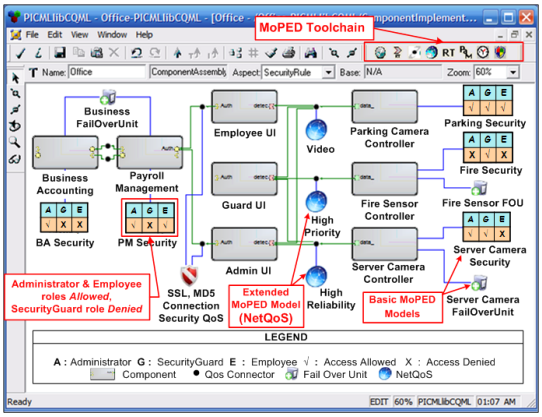

Fig. 6. **The MoPED model of the Office Enterprise case study**

A. Evaluating Reduction in Manual Efforts

To evaluate the modeling effort, we used the Platform-independent Component Modeling Language (PICML) [3] to model the functionality of the office enterprise and hazard sensing system in the case study and Lightweight CORBA Component Model (LwCCM) as our component-based implementation platform. PICML is a GME-based DSML that simplifies the composition of component-based systems.

Figure 6 shows 8 components and 10 connections in the case study system. For every component without any QoS attributes, the LwCCM descriptor format requires 17 lines of XML code, whereas for a connection it requires 14 lines of XML code. Without any dependability configuration properties, a model of our case study would require 8 * 17 + 14 * 10 = 276 lines of XML code. However, the size of descriptors grows rapidly as the number of dependability configuration properties increases.

Table I shows an increase in the generated lines of metadata as a result of associating exactly one QoS model among *FailOverUnit*, *ComponentSecurityQoS*, and *ConnectionSecurityQoS* to a component with a single connection. This table summarizes the smallest possible change in the size of metadata when a single modeling element is annotated with QoS. The figures in Table I are used as a scale to show the growth in the size of metadata for the case study model shown in Figure 6. Table II shows the number of lines of generated XML metadata for our model of the case study. The MoPED tool chain generates as many as 1,098 lines of additional metadata,

Without the automated support of MoPED, modelers would need to manually update metadata with availability, security, and network QoS requirements, which is tedious and error-prone. The MoPED tool chain automatically generates consistent metadata taking into account all three QoS requirements simultaneously, thereby simplifying modeling significantly.

VI. Related Work

Many techniques have been devised for model-based provisioning of computer system dependability. Previous research for dependability modeling and analysis, such as [6], [8], [27], [5], are based on OMG's Model Driven Architecture [18] (MDA) and use UML profiles to capture system dependability architectures. Model-to-Model (M2M) transformations are used for step-wise refinement of platform-independent models into platform-specific models (*e.g.,* Java code skeletons) or in to formal analysis models such as Fault Trees, Markov models, and Stochastic Petri-Nets (SPN). Finally, existing formal analysis tools are used to perform reliability analysis.

Lack of domain-specific semantics makes the above approaches difficult to use and understand. Moreover, the above UML profiles consider only one dimension of dependability at a time, limiting their applicability in the integrated availability/security approach. Our approach in MoPED is based on domain specific modeling, which provides high-level intuitive modeling abstractions to capture requirements. Moreover, requirements of availability and security can be simultaneously captured using unified QoS modeling framework of MoPED.

The OMG has adopted UML profile [19] for schedulability, performance and time specification and a more general profile [20] for modeling QoS. These UML profiles provide a way to specify the QoS ontology with QoS characteristics with support for attaching QoS requirements to the core UML diagrams. A common feature between these standard UML profiles and the modeling language of MoPED is the first class support for QoS concerns and a QoS modeling framework built around it. Our QoS modeling framework differs from that of

Metadata artifacts	FailOverUnit (replication degree = 1)	Component Security QoS	Connection Security QoS
New components	17	0	0
New connections	14	0	0
Component properties	0	11	0
Connection properties	0	0	11

TABLE I
Increase in the Number of Lines of Descriptors of a Single Component with a Single Connection

Metadata artifacts	FailOverUnit (replica = 1)	FailOverUnit (replica = 3)
New components	4	12
New component lines (* 17)	68	204
New connections	15	45
New connection lines (* 14)	210	630
	SecurityQoS	**SecurityQoS**
Component property lines (* 11)	44	132
Connection property lines (* 11)	44	132
Total number of lines	366	1098

TABLE II
Increase in the Number of Lines of Descriptors for the Model of the Case Study

the standard QoS profiles because it is tailored to the domain of component-based systems.

Early efforts on integrating security with availability and reliability were toward developing a unified terminology [10] to describe systems using an *extended* meaning of definitions of dependability attributes. A systemic conceptual model is suggested in which various aspects of security and dependability are discussed. Based on the unified terminology, several techniques for performing dependability evaluation can be applied in the security domain.

Nicol et al. [16] presents an excellent survey of existing model-based techniques for evaluating system dependability, and summarize how they are being extended to evaluate system security. In that respect, our work is complementary to the above more formal approaches. MoPED can be extended to capture formal semantics and transform them into analysis models using the MoPED tool chain. After validating system's QoS specification models using analysis tools, MoPED can be used to automatically transform the decisions into platform-specific deployment and configuration metadata, significantly reducing the manual efforts.

VII. Concluding Remarks

This paper described our approach to modeling key attributes of dependable systems (availability and security) via a unified QoS modeling framework. We presented our approach in the context of a model-driven tool chain called MoPED that provides intuitive, domain-specific modeling abstractions to capture availability, security QoS requirements of component-based systems. MoPED helps designers resolve potential conflicts between availability and security requirements.

We evaluated the capabilities of MoPED using a representative case study of an enterprise office and hazard sensing system that has simultaneous availability, security requirements. Our evaluation of MoPED indicates that it prevents designers from making mistakes in the QoS configuration and significantly simplifies system deployment by automating the generation of platform-specific metadata that faithfully reflects the dependability QoS decisions.

References

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems. In K. G. Larsen and P. Niebert, editors, *Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6-7, 2003. Revised Papers*, volume 2791 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2003.

[2] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability, 2001.

[3] 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 *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 190–199, Los Alamitos, CA, USA, 2005.

[4] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From uml models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, 2006.

[5] 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*, page 64, Washington, DC, USA, 1999. IEEE Computer Society.

[6] A. Capozucca, B. Gallina, N. Guelfi, P. Pelliccione, and A. Romanovsky. CORRECT - Developing Fault-Tolerant Distributed Systems. *ERCIM News*, 64(1), 2006.

[7] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, Massachusetts, 2000.

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

[9] J. H. Hill, S. Tambe, and A. Gokhale. Model-driven Engineering for Development-time QoS Validation of Component-based Software Systems. In *Proceedings of 14th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 07)*, pages 307–316, Tucson, AZ, Mar 2007.

[10] E. Jonsson. An integrated framework for security and dependability. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 22–29, New York, NY, USA, 1998. ACM.

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

[12] Z. Kalbarczyk, R. K. Iyer, and L. Wang. Application fault tolerance with armor middleware. *IEEE Internet Computing*, 9(2):28–37, 2005.

[13] A. Ledeczi, A. Bakay, M. Maroti, P. Volgysei, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *IEEE Computer*, pages 44–51, November 2001.

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

[15] 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*, 17(12):1527–1545, 2005.

[16] D. M. Nicol, W. H. Sanders, and K. S. Trivedi. Model-based evaluation: From dependability to security. *IEEE Trans. Dependable Secur. Comput.*, 1(1):48–65, 2004.

[17] OASIS eXtensible Access Control Markup Language (XACML) TC. eXtensible Access Control Markup Language TC v2.0 (XACML). http://www.oasis-open.org, 2005.

[18] Object Management Group. *Model Driven Architecture (MDA)*, OMG Document ormsc/2001-07-01 edition, July 2001.

[19] Object Management Group. *UML Profile for Schedulability, Performance, and Time Specification*, Final Adopted Specification ptc/02-03-02 edition, Mar. 2002.

[20] Object Management Group. *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Joint Revised Submission*, OMG Document realtime/03-05-02 edition, May 2003.

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

[22] J. Peleska. Uniform workbench - formal methods and the development of dependable systems.

[23] S. Tambe. *Model-driven Fault-tolerance Provisioning for Component-based Distributed, Real-time and Embedded Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, Oct. 2010.

[24] S. Tambe, J. Balasubramanian, A. Gokhale, and T. Damiano. MDDPro: Model-Driven Dependability Provisioning in Enterprise Distributed Real-Time and Embedded Systems. In *Proceedings of the International Service Availability Symposium (ISAS)*, Durham, New Hampshire, USA, 2007.

[25] S. Tambe, A. Dabholkar, and A. Gokhale. CQML: Aspect-oriented Modeling for Modularization and Weaving QoS Concerns in Component-based Systems. In *Proceedings of the* 16^{th} *IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 09)*, pages 11–20, San Francisco, CA, Apr. 2009. IEEE Computer Society.

[26] S. Tambe, A. Dabholkar, and A. Gokhale. Generative Techniques to Specialize Middleware for Fault Tolerance. In *Proceedings of the* 12^{th} *IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2009)*, Tokyo, Japan, Mar. 2009. IEEE Computer Society.

[27] A. Zarras, P. Vassiliadis, and V. Issarny. Model-Driven Dependability Analysis of Web Services. In *Proc. of the Intl. Symp. on Dist. Objects and Applications (DOA'04)*, Agia Napa, Cyprus, Oct. 2004.