Building Secure Systems Using Model-Based Engineering and Architectural Models

Dr. Jörgen Hansson, Dr. Peter H. Feiler, and John Morley
Software Engineering Institute

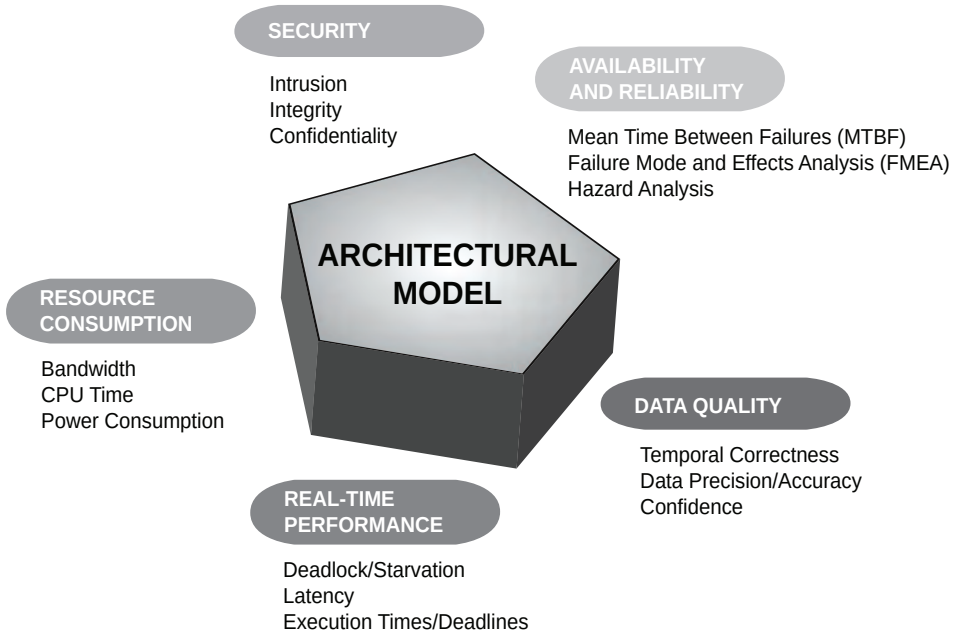
The Department of Defense's policy of multi-level security (MLS) has long employed the Bell-LaPadula and Biba approaches for confidentiality and integrity; more recently, the multiple independent levels of security/safety (MILS) approach has been proposed. These approaches allow designers of software-intensive systems to specify security levels and requirements for access to protected data, but they do not enable them to predict runtime behavior. In this article, model-based engineering (MBE) and architectural modeling are shown to be a platform for multi-dimensional, multi-fidelity analysis that is conducive for use with Bell-LaPadula, Biba, and MILS approaches, and enables a system designer to exercise various architectural design options for confidentiality and data integrity prior to system realization. In that way, MBE and architectural modeling can be efficiently used to validate the security of system architectures and, thus, gain confidence in the system design.

System designers face several challenges when specifying security for distributed computing environments or migrating systems to a new execution platform. Business stakeholders impose constraints due to cost, time-to-market requirements, productivity impact, customer satisfaction concerns, and so forth. Thus, a system designer needs to understand requirements regarding the confidentiality and integrity of protected resources (e.g., data). Additionally, a designer needs to predict the effect that security measures will have on other runtime quality attributes such as resource consumption, availability, and real-time performance. After all, the resource costs associated with security can easily overload a system. Nevertheless, security is often studied only in isolation and late in the process. Furthermore, the unanticipated effects of design approaches or changes are discovered only late in the life cycle when they are much more expensive to resolve[1].

MBE for Security Analysis

Modeling of system quality attributes, including security, is often done – when it is done – with low-fidelity software models and disjointed architectural specifications by various engineers using their own specialized notations. These models are typically not maintained or documented throughout the life cycle, making it difficult to obtain a system view. However, a single-source architecture model of the system that is annotated with analysis-specific information allows changes to the architecture to be reflected in the various analysis models with little effort; those models can easily be regenerated from the architecture model (Figure 1). This approach also allows the designer to conduct an adequate trade-off analysis and evaluate architectural variations prior to system realization, thereby gaining confidence in the architectural design. Models also can be used to evaluate the effects of reconfiguration and system revisions in post-development phases.

Figure 1: *A Single, System Architectural Model Annotated for Multiple Non-Functional Quality Analyses*



Using MBE tools, the Software Engineering Institute (SEI) has developed analytical techniques to:

- Represent standard security protocols for enforcing confidentiality and integrity, such as Bell-LaPadula [1, 2], Chinese wall [3, 4], role-based access control [5], and the Biba model [6].
- Model and validate security using system architecture according to flow-based approaches early and often in the life cycle.

The MBE tools that the SEI uses are the Architecture Analysis and Design Language (AADL) and the Open Source AADL Tool Environment (OSATE) set of analysis plugins [7][2]. The AADL is used to model and document system architecture and provide the following platform for analyses:

- Using a single architecture model to evaluate multiple quality attributes, including security.
- Early and often during system design or when upgrading existing system architecture.
- At different architecture refinement levels as information becomes available.
- Along diverse architectural aspects, such as behavior and throughput.

Architectural Considerations

Security as an architectural concern crosscuts all levels of the system (application, middleware, operating systems, and hardware). Thus, security requires intra- and inter-level validation and has immediate effects on the runtime behavior of the system, specifically on other dependability attributes.

The designer needs to enforce intra- and inter-level security throughout the architecture. Figure 2 depicts various system levels involved in the validation of security privileges against confidentiality requirements (it assumes that authentication and other necessary security services are enforced). The designer seeks to ensure that the software applications do not compromise the confidentiality of the secure information they are exchanging. Consequentially, software applications need to execute on top of a secure operating system, be mapped to a protected and secured hardware memory space, and communicate over a secure communication channel. If the data is labeled "confidential," then every architectural layer needs to have a clearance of at least that level.

Additionally, the designer needs to acknowledge that security comes with a cost. Encryption, authentication, security, and protection mechanisms increase bandwidth demand in terms of the central processing unit (CPU), the network, and memory. These increases affect the temporal behavior of the system (worst-case execution time, response time, schedulability, and end-to-end latency) as well as power consumption (especially important in battery-driven or limited lifetime devices such as sensor networks or portable communication devices).

As a result, security cannot be considered in isolation. The system designer makes choices to trade these quality attributes against each other (a particular concern for embedded and real-time systems, which operate under significant resource constraints while ensuring high levels of dependability and security). Security is interlinked with other non-functional behaviors such as predictability/timeliness and resource consumption, as well as inadvertent effects on reliability and availability. Figure 3 illustrates some of those dependencies on the single-model, multiple-analysis view.

An MBE Approach to Validating Confidentiality

Confidentiality addresses concerns that sensitive data should be disclosed to or accessed only by authorized users (i.e., enforcing prevention of unauthorized disclosure of information). Data integrity is closely related, as it concerns prevention of unauthorized modifications of data.

To model and validate the confidentiality of a system, we distinguish between general and application-dependent validation. General validation of confidentiality is the process of ensuring that a modeled system conforms to a set of common conditions that support system confidentiality independent of a specific reasoning framework for security. MBE takes advantage of the versatile concept of subjects operating on objects by permissible access (read, execute, append, and write), a notion introduced by Bell and LaPadula [1], enabling us to model and validate security at both the software and hardware levels.

This form of validation assumes that subjects and objects are assigned a security level that is the minimum representation to enforce basic confidentiality and need-to-know principles. By contrast, application-specific validation relies on detailed confidentiality requirements and a specific, reasoning-based security framework.

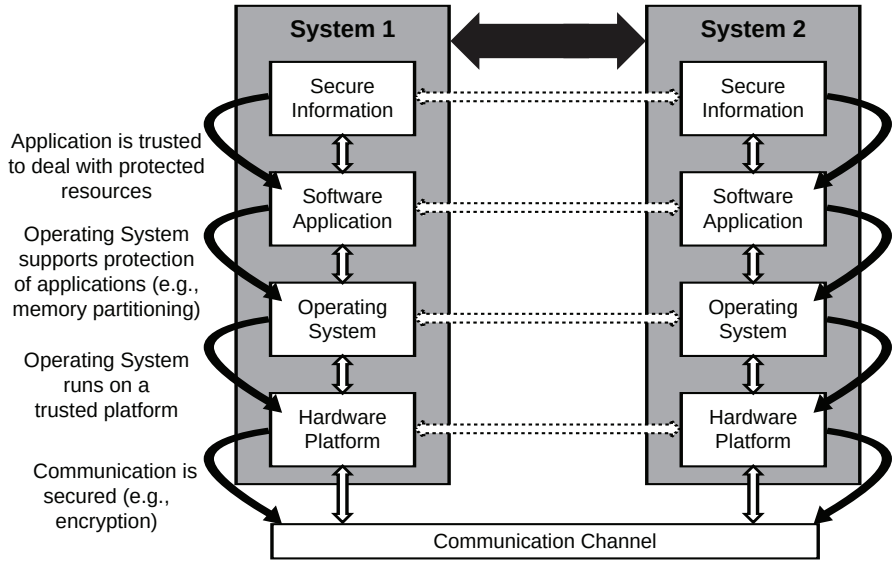


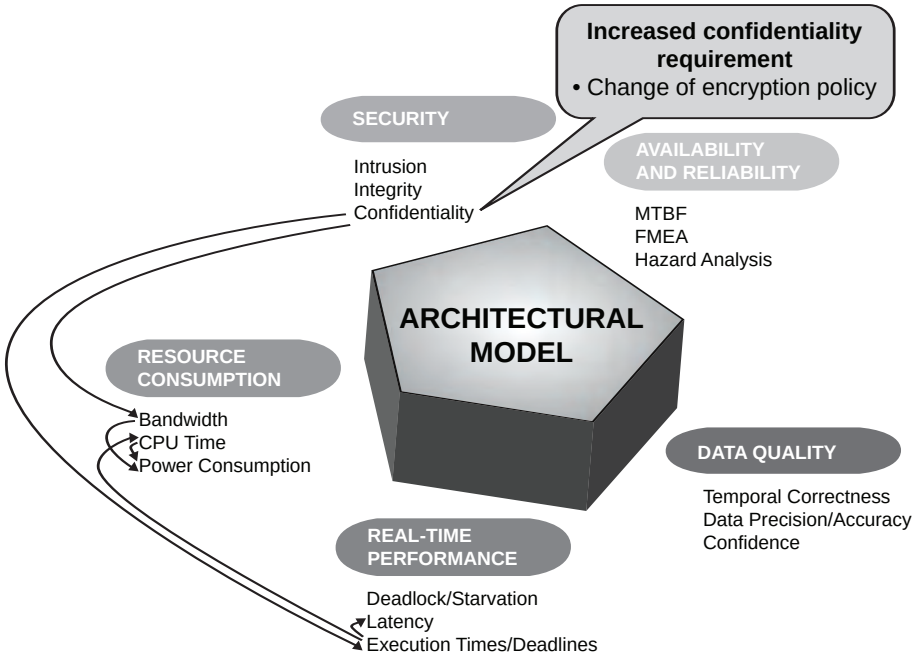
Figure 2: *System Perspective on Security*

The MBE security framework features:

- Representation of the confidentiality requirements of resources (i.e., objects).
- Representation and generation of security clearance/least privileges[3] of subjects operating on the objects.
- Representation of authorized operations, ensuring unauthorized infiltration, unauthorized exfiltration, and unauthorized median of actions. This is captured in an access matrix.

With the object's security requirements specified in an AADL model, the least amount of privileges for the subjects can be generated in a straightforward manner. Given that the subjects' privileges are specified, a mismatch between the least privilege and what has been specified

Figure 3: *Single Architectural Model Showing an Example of Impact on and Interaction of Non-Functional Behavior Due to a Change in Security*



```
-- Property intended to be customized by modelers.
-- Parameterizes the security property definitions.
property set Security_Types is
   -- Military levels by default
   Classifications:
     Type enumeration (unclassified, confidential, secret,
                          top_secret);
   -- This must be the first element of Classifications
   Default_Classification:
     constant Security_Types::Classifications =>

   -- Default set of categories
   Categories:
      type enumeration (A, B, C, D);
end Security_Types;
```

Figure 4: *Specification of Security Levels*

means the assigned privilege is either insufficient or greater than the minimum privilege. The latter result may be unnecessary or an indication that the subject might be associated with objects not yet described in the model.

The following types of security validation and analysis are available as OSATE plugins:

- **Basic confidentiality principle.** Access should only be granted if given the appropriate security clearance.
- **Need-to-know principle.** Access should be granted to a resource only if there is a need.
- **Controlled sanitization.** Lowering the security level of an object or subject should only be authorized and performed by a privileged subject.
- **Non-alteration of object's security requirements.** A subject using an object as input should not alter the security level of the object, even if the object is updated as an output from the subject.
- **Hierarchical conditions.** A component has (1) a security level that is the maximum of the security levels of its subcomponents, and (2) all connections are checked to determine whether the source component of a connection declaration has a security level that is the same or lower than that of the destination component.

Using OSATE and the AADL, system designers and developers can add analysis techniques as needed.

The validation through architectural modeling of system security – given the confidentiality requirements of data objects and the security clearance by users – must include validation of (1) software architecture, and (2) system architecture where the software architecture is mapped to hardware components.

By mapping the entities of a software architecture (e.g., processes, threads, and partitions) to a hardware architecture (consisting of, for example, CPUs, communication channels, and memory), we can ensure that the hardware architecture supports required security levels, as described in Figures 4 and 5.

Consider the scenario of two communicating processes, both requiring a high level of security because the data objects require secret clearance. The system platform in this scenario consists of a set of CPUs with hardware support for various algorithms that encrypt messages before network transmission. By modeling the system, we can represent and validate that both processes and threads (now considered to be objects) can be executed (access mode) on CPUs (subjects) with adequate encryption support. Furthermore, we can validate that CPUs (objects) communicate data (access modes of writing and reading) over appropriately secured communication channels (subjects). In a similar fashion, we can enforce design philosophies saying that only processes of the same security level are allowed to co-exist within the same CPU or partition, or that they can write to a secured memory.

A combination of the AADL and the OSATE security plugin tool has been put into use in industry. Rockwell-Collins used the technology to enable the high-assurance handling of data from multiple sensors having varying levels of security, such as airborne imagery with the Field Programmable Gate Array (FPGA). Typically, a high-assurance processor is used to securely tag variable input. An FPGA is powerful and fast. It is deemed easier to develop applications on an FPGA, which also reduces the cost and time-to-market. Furthermore, the FPGA can be reprogrammed at runtime (e.g., to fix bugs), which can lower maintenance-engineering costs. Because FPGA behavior is more complex, architecture-level definition and analysis are needed. To this end, Rockwell-Collins developed architectural models of the FPGA using AADL and used the OSATE tool to validate security and demonstrate the high-assurance potential of FPGAs.

Figure 5: *Architectural Components to Which Security Levels and Requirements Can Be Connected*

```
property set Security_Attributes is
  Class: inherit Security_Types::Classifications =>
    value(Security_Types::Default_Classification)
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);

  Category: inherit list of Security_Types::Categories =>
    ()
    applies to (data, subprogram, thread, thread group,
                process, memory, processor, bus, device,
                system, port, server subprogram,
                parameter, port group);
  -- . . .
end Security_Attributes;
```

Validating MILS Architectures With the MBE Approach

The AADL and OSATE tools can be used to validate the security of systems designed using the MILS[4] architecture approach (see [8, 9]). MILS uses two mechanisms to modularize – or divide and conquer – in architecting secure systems: partitions, and separation into layers. The MILS architecture isolates processes in partitions that define a collection of data objects, code, and system resources and can be evaluated separately. Each partition is divided into the following three layers, each of which is responsible for its own security domain and nothing else:

1. **Separation Kernel (SK).** Responsible for enforcing data isolation, control of information flow, periods processing,

and damage limitation. An example is the SK Protection Profile [8].

2. **Middleware service layer.**
3. **Application layer.**

Thus, the MILS separates security mechanisms and concerns into the following three component types, classified by how they process data:

- **Single-Level Secure (SLS).** Processes data at one security level.
- **Multiple Single-Level Secure (MSLS).** Processes data at multiple levels, but maintains separations between classes of data.
- **MLS.** Processes data at multiple levels simultaneously and transforms data from one level to another.

The strength of the MILS architecture lies in its reductionist approach to decompose a system into components of the above-mentioned types that would be more manageable to certify. These components are also mapped to partitions (and, as mentioned earlier, the MILS architecture approach builds on partitioning as one key concept to enforce damage limitation and separation of time and space).

An MBE approach is conducive to the validation concerns most critical to MILS, including:

- **Validating the structural rigidity of architecture, such as the enforcement of legal architectural refinement patterns of a security component into SLS, MSLS, and MLS types.** Given that an MILS architecture design and system is decomposed into security components that can be certified in isolation, the structural rigidity concerns the legal mappings and connections of the components. The decomposition into SLS, MSLS, and MLS types can be applied to components, connectors, and ports. Furthermore, each component can be divided into parts using the product, cascade, or feedback decomposition patterns [10, 11, 12]. For example, an MSLS component with *n* security levels can be decomposed into *n* distinct SLS components. Thus, confidence in the validation of an architecture increases with the fidelity of the modeling. By using an architectural model in AADL to capture the security types and multiple architectural levels, MBE analysis is conducted to validate the correctness of the decompositions and mappings.
- **Architectural modeling and validation of assumptions underlying MILS.** Fundamental to enforcement of security in an MILS architecture is having a system that supports partitioning, specifically damage limitation and separation in time and space. By partitioning the system, one minimizes the risk of illegal component interactions among components and protects components from the faulty behavior. This can be realized in the system architecture by ensuring fault-containment and deploying security-cognizant memory allocation so that MILS components and tasks reside in protected memory spaces – and do not co-reside in the same memory space if they differ in security levels. Similarly, separation in time can be ensured through avoiding the interleaved execution of tasks with different security levels, realized in partition scheduling and validating execution behaviors. The AADL supports the modeling of partitions and virtual processors. As well, the virtual machine mechanism is recognized as a key concept for providing robustness through fault containment because it provides time and space partitioning to isolate application components and subsystems from affecting each other (due to sharing of resources). This architecture pattern can be found in the Aeronautical Radio Incorporated (ARINC) 653 standard [13]. A single-source architectural model in AADL can thus be used to validate the security requirement in an architectural context, specifically the MILS composition, and the architectural assumptions required.
- **Validating requirements specific to the NEAT characteristics and the communication system.** MILS requires that its SK and the trusted components of middleware services are implemented so that the security capabilities enforce what is commonly referred to as the NEAT characteristics:
  - **N***on-bypassable*. Security functions cannot be circumvented.
  - **E***valuatable*. The size and complexity of the security functions allow them to be verified and evaluated.
  - **A***lways* invoked. Security functions are invoked each and every time without exception.
  - **T***amperproof*. Subversive code cannot alter the function of the security functions by exhausting resources, overrunning buffers, or other forms of making the security software fail.

The MBE approach allows designers to assure that software applications execute on top of a secure operating system, map to a protected and secured hardware memory space, and communicate over secure communication channels. It also enables the analysis of security measures early and throughout the development life cycle.

Conclusions

The objective of a secure system implies that security clearances are given conservatively. The MBE approach supports this objective through enabling analysis of the architectural model to derive the minimum security clearance on components. By providing mechanisms to ensure that sanitization is conducted within allowed boundaries, the MBE approach enables the system designer to analyze and trace more threatening security risks, since sanitizing actions are permitted exemptions of security criteria and rules, and as such should be minimized in the system.

Security analysis using the MBE approach also supports:

- The evaluation of an architecture configuration with respect to impact on other non-functional attributes, such as increases in power consumption, bandwidth usage, and performance.
- The validation of architectural requirements necessary to enforce the MILS approach to containing faults, through partitioning and separation in time and space.
- A reduction of the effort necessary for re-certification in the event of architectural changes.

Furthermore, validation of security can be conducted at multiple layers and different levels of fidelity, early and throughout the development life cycle.◆

References

1. Bell, D.E., and L.J. LaPadula. Secure Computer Systems: Mathematical Foundations MITRE Technical Report 2547, Vol. 1. Bedford, MA: MITRE Corporation, 1973 <www.albany.edu/acc/courses/ia/classics/belllapadula1.pdf>.
2. Bell, D.E., and L.J. LaPadula. Secure Computer Systems: Unified Exposition and MULTICs Interpretation MITRE Technical Report ESD-TR-75-306. Bedford, MA: MITRE Corporation, 1976 <http://csrc.nist.gov/publications/history/bell76.pdf>.
3. Brewer, David D.C., and J. Michael Nash. "The Chinese Wall Security Policy." IEEE Symposium on Security and Privacy. Oakland, CA: 1-3 May 1989 <www.gammassl.co.uk/topics/chinesewall.html>.
4. Lin, T.Y. Chinese Wall Security Policy – An Aggressive Model. Proc. of the Fifth Aerospace Computer Security Application Conference. Tucson, AZ: 4-8 Dec. 1989 <http://ieeexplore.ieee.org/iel5/7100/19131/00884701.pdf>.

5. Ferraiolo, David, and Rick Kuhn. Role-Based Access Control. Proc. of the 15th National Computer Security Conference. Baltimore, MD: 13-16 Oct. 1992 <http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf>.
6. Biba, K.J. Integrity Considerations for Secure Computer Systems MITRE Technical Report-3153. Bedford, MA: MITRE Corporation, Apr. 1977.
7. AADL. Find and Solve Problems Before Runtime With Model-Based Engineering. 14 Apr. 2008 <www.aadl.info>.
8. The Common Criteria Evaluation and Validation Scheme. Validation Protection Profile – U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness, Vers. 1.03. 11 July 2008 <www.niap-ccevs.org/cc-scheme/pp/id/pp_skpp_hr_v1.03>.
9. Alves-Foss, J., W.S. Harrison, P. Oman, and C. Taylor. "The MILS Architecture for High-Assurance Embedded Systems." International Journal of Embedded Systems 2.3/4 (2006): 239-347.
10. Zhou, J., and J. Alves-Foss. "Security Policy Refinement and Enforcement for the Design of Multi-Level Secure Systems." Journal of Computer Security 16.2 (2008): 107-131.
11. McLean, J. "Security Models." Encyclopedia of Software Engineering. John Wiley & Sons, New York, NY: 1994.
12. Zakinthinos, A. "On the Composition of Security Properties." Diss. U of Toronto, Mar. 1996.
13. ARINC Incorporated. Avionics Application Software Standard Interface, ARINC 653 Standard Document 14 Apr. 2008 <www.arinc.com>.
14. National Institute of Standards and Technology (NIST). The Economic Impacts of Inadequate Infrastructure for Software Testing. NIST Planning Report. May 2002 <www.nist.gov/director/prog-ofc/report02-3.pdf>.

Notes

1. A NIST study observed that 70 percent of all defects are introduced prior to implementation. Yet only 3.5 percent of the defects were detected in these phases, while 50.5 percent of the faults were detected in the integration phase. The defect removal cost ranged from 5 to 30 times relative to the cost of removing the defect in the phase of introduction (if it had been detected). Other sources are reporting similar estimates; while the numbers vary, the conclusions do not [14].
2. The AADL, an international industry standard, incorporates an XML/XMI exchange format to support model interchange and tool chaining. AADL also can be used (1) with UML state and process charts through its UML profile, (2) to drill into root causes and develop quantitative analysis as a follow-up to the SEI Architecture Tradeoff Analysis Method®, and (3) in conjunction with assurance cases, to support claims made about the safety, security, or reliability of a system. The freely available OSATE includes analysis plugins for performance, resource consumption, security, and reliability.
3. The principle of least privilege has been identified as important for meeting integrity objectives; it requires that a user (subject) be given no more privilege than necessary to perform a job. This principle includes identifying what the subject's job requires and restricting the subject's ability by granting the minimum set of privileges required.
4. MILS has been proposed as an approach to building secure systems [9, 10]. MILS is a joint research effort of academia, industry, and government, led by the U.S. Air Force Research Laboratory. The MILS approach is based on the notion of separating – and thus limiting the scope and reducing the complexity of – the security mechanisms.

® The Architecture Tradeoff Analysis Method is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

About the Authors

Jörgen Hansson, Ph.D., is a senior member of the technical staff and lead of the Performance-Critical Systems (PCS) Initiative at the SEI, and is a professor of computer science at Linköping University, Sweden. The Initiative focuses on developing, maturing, and transitioning analysis-based assurance and model-based engineering tools and practices for predicting the dependability and performance of software systems. Hansson holds bachelor's and master's degrees in computer science from University of Skövde, Sweden, and a doctorate in computer science from Linköping University, Sweden.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-6733
Fax: (412) 268-5758
E-mail: hansson@sei.cmu.edu

Peter H. Feiler, Ph.D., is a senior member of the technical staff in the PCS Initiative at the SEI. He has authored more than 80 articles in the areas of dependable real-time systems, architecture languages for embedded systems, and predictable system analysis and engineering. Feiler is the technical lead and author of the SAE AS-2C AADL standard. Feiler holds a Vordiplom degree in math/computer science from Technical University Münich, and a doctorate in computer science from Carnegie Mellon University.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-7790
Fax: (412) 268-5758
E-mail: phf@sei.cmu.edu

John Morley is a member of the operating staff at the SEI. He has reported on model-based engineering and service-oriented architecture for SEI publications, has more than 20 years experience in writing and editing scientific and technical materials, and holds a master's degree in English literature from Duquesne University.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-6599
Fax: (412) 268-5758
E-mail: jmorley@sei.cmu.edu