Toward Model-Based Embedded System Validation through Virtual Integration

by Peter H. Feiler and Jörgen Hansson

Introduction
Embedded systems are safety-critical and mission-critical systems that have become increasingly software-intensive - with millions of lines of code executing on a distributed networked set of processors. Developing such systems has indeed shown to be increasingly challenging as embedded software subsystems compete for computer system resources and face unpredictable interaction behavior due to the time-sensitive nature of the application. Several studies [1,2,3] have shown that 70% of faults are introduced early in the life cycle, while 80% of them are not caught until integration test or later with a repair cost of 16x or higher. The diagram illustrated in Figure 1 shows percentages for fault introduction, discovery, and cost factors. The cost of developing the software for an aircraft has become unaffordable – reaching $10B or more. If we can discover a reasonable percentage of these late system-level faults earlier in the development process we can expect considerable cost savings.

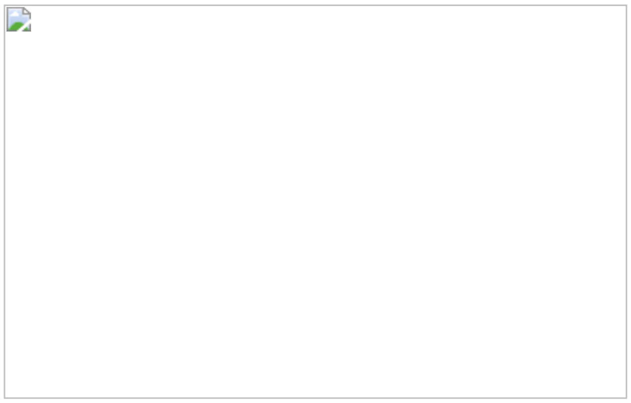

Figure 1: Fault Introduction, Discovery and Cost Factors

There are many cases of systems failing due to a software error despite the fact that the best design techniques and fault-tolerance techniques are being used. [7] examines some of them and the root causes due to mismatched assumptions. A well-publicized example of a system failure due to software is the explosion of the Ariane 5 rocket during her maiden flight. The destruction was triggered by the overflow of a 16-bit signed integer variable due to Ariane 5´s greater acceleration in a reused Ariane 4 software component to perform a function that was “not required for Ariane 5”. This fault was not caught because the handler was disabled due to efficiency considerations and cascaded into a total system failure [8]. Even simple systems such as the ITunes music program failed when it was migrated to a dual-core processor system. Two tasks were involved in ripping CDs. When executing on a single processor first one task and then the other task was performed. Once on a dual-core processor, the two tasks executed at the same time and got in each other´s way updating the music catalog [9]. Finally, with regard to the Mars Pathfinder after a successful landing, not long after it started gathering meteorological data the spacecraft began experiencing system resets. The press reported these failures in terms such as "software glitches" and “the computer was trying to do too many things at once”. An analysis on the ground identified the problem as priority inversion due to a low priority task blocking a high priority task on a lock to a shared data area. Once recognized the engineers could remotely enable the use of a priority ceiling protocol that overcomes the inversion problem and the mission completed successfully [10].

Quantitative architectural modeling, verification and validation of software and system behaviors have shown significant promise in addressing defects related to requirements and design shortcomings for embedded software.

Side bar 1:
A number of recent studies confirm that a paradigm shift towards analysis and formal validation at the architecture level must occur to meet the challenges of these time-sensitive software-reliant systems with high safety and reliability demands:
GAO Space-based SW Study [11] highlighting the reality of more testing than planned (exhausting vs. exhaustive testing) due to the increasingly complex interactions between system components;
NASA Software Complexity Study [12] on flight software growth & complexity, and the need for integration of fault prevention, detection, and containment with nominal system operation;
Leveson System Safety Engineering Study [13] on accident model based systems theory focusing on accident factors (understanding of root causes);
National Research Council Study [14] by the committee for certifiably dependable software systems addressing the issue of sufficient evidence for software for dependable systems through analysis and formal validation.

The Promise of Model-based Engineering
Model-based engineering (MBE) involves the creation of models of the system with focus on a certain aspect of the system such as functionality, performance, or reliability. Modeling, analysis and simulation has been practiced by engineers for a number of years. Traditionally, this has led to the

January 2010
Vol. 12, Number 4

Model-Driven Development

Articles in this issue:
Tech Views
Domain-Specific Modeling for Full Code Generation
Model-Based Testing - Next Generation Functional
Theory-Practice-Tools for Automated Statistical
Toward Model-Based Embedded System Validation through Virtual Integration
Understanding Temporal Logic -The Temporal Engineering of Software

Share

Comment

Download this issue (PDF)

How did you rate this issue?

Receive the Journal of Software Technology

Click here to submit
an article or to check out future themes of the Journal of Software Technology

+ Feedback

STN Issues
2011
2010
13-3 - Welcome To The New DACS!
13-2 - Defensive Cyber Security
13-1 - Software Quality, Reliability, an...
12-4 - Model-Driven Development
2009
2008
2007
2006
2005
2004
2003
2002
2001
2000
1999
1998
1997
1996
1995
1994
1993

About the Journal of Software Technology

Advertising Opportunities

Article Reprints

creation of different models of the same system to represent the interests of different stakeholders. For example, computer hardware models have been created in Very High Speed Integrated Circuits Hardware Description Language (VHDL) [15] and validated through model checking [16]. Control engineers have used modeling languages such as Simulink for years to represent the physical characteristics of the system to be controlled and the behavior of the control system. Characteristics of physical system components, such as thermal properties, fluid dynamics, and mechanics, have been modeled and simulated. Dependability engineering resulted in the creation of fault trees for fault analysis and Markov models for reliability analysis. Resource utilization analysis is based on resource demands and resource capacities. Scheduling analysis is based on a timing model of the application tasks. Security analysis involves the creation of a model in terms of security levels and domains applied to subjects and objects.

Despite these modeling efforts system-level problems remain late into the development life cycle. These different models are created at different times by different stake holder teams interpreting architecture & design documents to gain their understanding of the system. These independently created models may not be consistent with each other and with the actual system as it evolves. Changes to the system do not always communicated to the various modeling teams and the various models become out of date. The result is a multiple truth problem that industry has experienced with model-based engineering. The analysis results have limited value as they reflect inconsistent and out of date models.

This traditional use of model-based development is understandable, given the lack of a common, precise architecture description language to represent the system architecture. To illustrate the need for an architecture model, consider the complexities of just one quality attribute – security. A system designer faces 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 the like. A system designer needs to understand requirements for the confidentiality and integrity of protected resources (e.g., data) and identify how the application software executing on a distributed computer system and interfacing with physical systems and human operators meets those requirements. In addition they have to predict the effect that security measures will have on other runtime quality attributes such as resource consumption, availability, and real-time performance. However, despite these considerations, security is often studied only in isolation and late in the process. The unanticipated effects of separated design approaches or changes are discovered only late in the life cycle, when they are much more expensive to resolve.

Only in the best case do the independent models created in the traditional approach reflect the same system architecture. Any change to the architecture during its lifetime requires each model to be updated and verified. As challenging as that is, the need to consistently proliferate changes in one analysis to others adds difficulty (e.g., reflect the impact of choosing a different security encryption scheme on intrusion resistance as well as on schedulability and end-to-end latency). Consequently, it has become important to move from the traditional approach toward the integration of the different analysis dimensions into a single, precisely specified architecture model. Creating a single architecture model of the system that is annotated with analysis-specific information can drive model-based development. It allows analysis-specific models to be generated from the annotated model and the regeneration with little effort of those models to reflect changes to the architecture. This new approach also allows the impact of changes across multiple analysis dimensions to be readily analyzed, giving the system designer more insight and the system integrator warning of costly side effects. Even after the development phases have been completed, the model (and its associated analysis models) can be used to evaluate effects of reconfiguration and system revisions.

Modeling the Embedded System Architecture with AADL

To precisely specify architecture, allow non-ambiguous interpretation, support quantitative analysis, and model embedded software-intensive systems, we need a common, standard language with strong semantics that can capture both the structure and dynamics of embedded systems. The requirements of such specification include: support for analyzable system properties to evaluate critical quality attributes; incremental modeling to support all life-cycle phases from early abstraction to final implementation; and evolution compositional representation to support hierarchical and abstract layering, allowing multiple levels of integration with suppliers and providing integrators with component integration extensibility to allow new representations of system behaviors as analysis approaches differ and grow.

As the importance of architecture has been recognized, industrial standards for architecture modeling have emerged: OMG SysML [6] for system engineering and SAE Architecture Analysis and Design Language (AADL) [4-5] for embedded software systems. SysML is a graphical modeling language in the form of an extensible Unified Modeling Language (UML) profile used early in system development to represent the requirements, component structure, discrete state behavior, and parametrics (i.e., physical behavior as equations) of a system. AADL is an international industry standard extensible architecture modeling language for embedded software systems with graphical and textual representations, well-defined execution semantics, and an XML interchange format.

The focus of AADL is to model the software system architecture in terms of an application system bound to an execution platform. Most of the modeling effort is directed to the application-specific software, incorporating sufficient operating system specifics to ensure that the model is adequate (precise, complete, and realistic) to support evaluation of a system architecture and verification. Many operating system characteristics are already embedded in AADL semantics to support architectural modeling (communication mechanisms, thread behavior, etc.). It is designed to support an architectural MBE development life cycle, including system specification, quantitative analysis, system tuning, efficient integration, and upgrade, and the integration of multiple forms of analyses and extensibility for additional analysis approaches.

AADL supports component-based modeling of the architecture. Components have a type and one or more implementations. Software components include data (to represent data types and shared data areas), subprogram, thread (to represent concurrently executing tasks), and process (to represent protected address spaces). The hardware components include processor, virtual processor (to represent schedulers and partitions), memory, bus, virtual bus (to represent virtual channels and protocols), and device (to represent physical system components). The system component is used to describe hierarchical grouping of components, encapsulating software components, hardware components and lower level system components within their implementations.

Interfaces to components and component interactions are completely defined. The AADL supports data and event flow, synchronous call/return and shared access. In addition it supports end-to-end flow specifications that can be used to trace data or control flow through components.

The core language supports modeling in several architectural views and addresses quality attribute analyses through explicit modeling of the application system and binding to execution platform components. Real-time analyses are supported through well-defined timing semantics for concurrent execution and interaction between the components, which are expressed through a hybrid automata specification in the standard document.

The AADL supports real-time task scheduling using different scheduling protocols. Properties to support General Rate Monotonic Analysis and Earliest Deadline First and other scheduling protocols are provided in the core standard. Execution semantics are defined for each category of component and specified in the standard with a hybrid automata notation.

Modal and configurable systems are supported by the AADL. Modes specify runtime transitions between statically known states and configurations of components, their connections and properties. Modes can be used for fault tolerant system reconfigurations affecting hardware and software as well as software operational modes.

The AADL supports component evolution through inheritance, allowing more specific components to be refined from more abstract components. Large scale development is supported with packages which provide a name space and a library mechanism for components, as well as public and private sections. Packages support independent development and integration across contractors.

AADL language extensibility is supported through a property sublanguage for specifying or modifying AADL properties, which can be used for additional forms of analysis. The AADL also provides an annex extension mechanism that can be used to specify sub-languages that will be processed within an AADL specification. An example is the Error Modeling Annex which allows specification of error models to be associated with core components. The combination of annex extensions and user defined property sets provide the means to integrate new specification capabilities and new analysis approaches to support the analysis tools and methods desired for multiple dimensions of analysis.

Safety-criticality is supported by AADL in a number of ways.

- AADL is strongly typed, e.g., if a processor specification indicates that it requires access to a Peripheral Component Interconnect (PCI) bus and an Ethernet, then only a PCI bus can get connected to the one bus access feature.
- The protected address space enforcement of processes and resource allocation enforcement of virtual processors ensure time and space partitioning.
- A safety level property on system components, initially used for major subsystems and later attached to more detailed components, is used to ensure that components with high safety criticality are not controlled by or receive critical input from low criticality components.
- AADL has a built-in fault-handling model for application threads and extends into an explicit representation of a health monitoring architecture, and fault management by reconfiguration, which is modeled through AADL modes.
- AADL supports fault modeling through the Error Model Annex standard, which allows us to introduce intrinsic faults, error state machines, and fault propagations across components – including stochastic properties such as probability of fault occurrence. These annotations to the architecture support hazard and fault impact analysis as well as reliability and availability analysis.
- The dynamic behavior of the architecture is represented by modes and further refined through the Behavior Annex standard of AADL. This allows us to apply formal methods such as model checking to validate system behavior.

Virtual System Integration and Validation
A key concept of virtual integration is the use of an annotated architecture model as the single source for architecture analysis. Thus, it becomes important to integrate the different analysis dimensions into a single architecture model. An architecture model that is annotated with analysis-specific information can drive model-based engineering by generating the analysis-specific models from this annotated model. This allows changes to the architecture to be reflected in the various analysis models with little effort by regenerating them from the architecture model. This approach also allows us to evaluate the impact across multiple analysis dimensions. In other words (see Figure 2), analytical models are auto-generated from an architecture model with well-defined semantics and annotated with relevant analysis-specific information (e.g., fault rates or security properties). Any changes to the architecture throughout the life cycle are reflected in all dimensions of analysis (e.g., substitution of a faster processor to accommodate a high workload not only is reflected in schedulability analysis, but also may impact end-to-end response time, and requires revalidation of increased power consumption against capacity as well as possible change in mass).

Figure 2: Analytical Models from an Annotated Architecture Model

A second key concept of virtual integration is the ability to interchange architecture models with their annotations through a standardized interchange format, such as the AADL XML standard. This allows system integrators and suppliers to exchange specifications and models of subsystems in a standardized representation and facilitates the integrator to routinely predict and validate system properties early and throughout the development life cycle.

Virtual integration facilitates quantitative analysis, verification and validation using models. As such it complements system testing by focusing on system issues due to concurrent execution and non-deterministic interaction of time-sensitive nature. To this end, AADL is being embraced by the V&V (validation and verification) community because it is a vehicle to express high-level system requirements and perform verification across a range of analyses. These analyses are made available through a number of tools such as the Eclipse-based open source OSATE, the commercial Stood environment [20], and tool chains put together in industry initiatives such as ASSERT [18], TOPCASED [17], and SPICES [19].

The Aerospace Vehicle Systems Institute (AVSI), a global cooperative of aerospace companies, government organizations, and academic institutions, has launched the System Architecture Virtual Integration (SAVI) program, which is a multiyear and multimillion dollar effort focused on developing a capability for system validation and verification through virtual integration of systems. Major players of the SAVI project include Boeing, Airbus, Lockheed Martin, BAE Systems, Rockwell Collins, GE Aviation, FAA, U. S. Department of Defense (DoD), Carnegie Mellon Software Engineering Institute (SEI), Honeywell, Goodrich, United Technologies, and NASA. The SAVI paradigm necessitates:

- An architecture-centric, multi-aspect model repository as the single source of truth
- A component-based framework to support model-based and proof-based engineering
- A model bus concept for consistent interchange of models between repositories and tools
- An architecture-centric acquisition process throughout the system life cycle that is supported by industrial standards and tool infrastructure

To establish cost-effective management and limit risks of the SAVI program, the Proof-Of-Concept (POC) project [21] has been executed as the first of multiple phases with the following goals:

- Document the main differences between a conventional acquisition process and the projected SAVI acquisition process and identify potential benefits of the SAVI acquisition process
- Evaluate the feasibility and scalability of the multi-aspect model repository and model bus concepts central to the SAVI project
- Assess the cost, risk, and benefits of the SAVI approach through a return on investment (ROI) study and development of a SAVI development roadmap

Together with the successful proof-of-concept demonstration of SAVI, an analysis to determine the economic gains was conducted, comparing the current acquisition and development practice (referred to "as-is") with a model-driven acquisition and development approach ("to-be"). Conservative estimates show that a cost reduction between 7.6%-26.1% is to be expected, which translates into a cost avoidance of $717M-$2,391M; the range is a function of the various levels of rework cost and anticipated defect removal efficiency.

Sidebar #2

Virtual System Integration Case Study by AVSI

For the proof of concept phase of the System Architecture Virtual Integration (SAVI) initiative, a multi-tier aircraft model was created and analyzed along multiple quality dimensions. The demonstration illustrated analyses at the Tier 1 level for system properties such as mass and electrical power consumption, and at the Tier 2 level focusing on the Integrated Modular Avionics (IMA) architecture by revisiting the previous analyses and adding computer resource analyses and end-to-end latency analysis across subsystems. The demonstration continued by showing AADL support to manage subcontracting with suppliers through a model repository. A negotiated subsystem specification could be virtually integrated and inconsistencies between supplier specifications such as inconsistent data representation, mismatched measurement units, and mapping into the ARINC 429 protocol could be detected. Three of the suppliers then developed a task-level specification of their subsystem and performed local validation through analysis. In one case, the subsystem was elaborated into behavioral specifications via UML diagrams, and an implementation in Ada. During this process the suppliers routinely delivered AADL models back to the airframer for repeated revalidation through virtual integration with increasing model fidelity. The demonstration model itself was developed by a team located in Iowa, Pennsylvania, France, and the UK utilizing the model repository located in Texas. At the end of the POC demonstration the SAVI team concluded that "the results of the demonstrations indicate that the SAVI concept is sound and should be implemented with further development".

Summary
In this article we have discussed architectural modeling, verification and validation as a means to realize virtual integration. The approach builds on AADL, which was designed for modeling software-intensive systems with real-time, embedded, fault tolerant, secure, safety-critical behaviors. Using the precise semantics of AADL, developers of subsystems can more readily share and understand architecture specifications. The language is designed to create a machine-analyzable, single architectural model annotated with precise notation. The language supports model creation and analysis throughout a full model-based development life-cycle .75 including system specification, analysis, system tuning, integration, and upgrade. There are several advantages with this approach. Virtual integration activities replace traditional design reviews by:

- Recording subsystem requirements in an initial system model during request for proposals
- Validating supplier model compatibility and initial resource allocations during proposal evaluation
- Validating interfaces and functionality during preliminary design integration
- Validating performance during critical design integration

Conducting early and continuous virtual model integration based on standardized representations has a number of advantages:

- It ensures that errors are detected as early as possible with minimal leakage to later phases
- Models with well-defined semantics facilitate auto-analysis and generation to identify and eliminate inconsistencies
- Automated compatibility analyses at the architecture level scale easily
- Industrial investment in tools is leveraged through well-defined interchange formats

About the Authors

Peter Feiler is a senior member of the technical staff at the Software Engineering Institute (SEI). He is the recipient of Carnegie Mellon´s 2009 Information Technology Award for his work on the Society of Automobile Engineers (SAE) standard Architecture Analysis and Design Language (AADL). He joined the SEI in 1985 after a successful career at Siemens Corporation, where he conducted research and led a group in software technology.

Jorgen Hansson is a senior member of the technical staff at the Software Engineering Institute (SEI) and a full professor of software engineering at Chalmers University of Technology, Sweden. His current research interests include embedded real-time systems, architectural design and validation, and dependability in terms of performance, availability, reliability, and security.

OSATE is the open-source AADL tool environment (OSATE) that includes an AADL front-end and architecture analysis capabilities as plug-ins to the open source Eclipse environment.

Author Contact Information
Email: **Peter Feiler**: lphf@sei.cmu.edu
Email: **Jorgen Hansson**: hansson@sei.cmu.edu

References

[1] The Economic Impacts of Inadequate Infrastructure for Software Testing (NIST Planning report 02-3, May 2002). Available: http://www.nist.gov/director/prog-ofc/report02-3.pdf. [Accessed: May 25, 2009].
[2] D. Galin, Software Quality Assurance: From Theory to Implementation. Pearson/Addison-Wesley, 2004.
[3] J. B. Dabney, “Return on Investment of Independent Verification and Validation Study Preliminary Phase 2B Report,” NASA IV&V Facility, 2003.
[4] SAE International. SAE Standards: Architecture Analysis & Design Language (AADL), AS5506, November 2004. Available: http://www.sae.org/technical/standards/AS5506/1. [Accessed May 25, 2009].
[5] P. H. Feiler, D. P. Gluch, and J. J. Hudak. “The Architecture Analysis & Design Language (AADL): An Introduction,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (USA), CMU/SEI-2006-TN-011, 2006.
[6] Systems Modeling Language (SysML). www.sysml.org .
[7] Peter H. Feiler, “Challenges in Validating Safety-Critical Embedded Systems”, Proceedings of SAE International AeroTech Congress, Nov 2009.
[8] “Ariane 5 Flight 501.”, en.wikipedia.org/wiki/Ariane_5_Flight_501.
[9] ITunes Crashes on Dual-core Processors. discussions.apple.com/thread.jspa?messageID=1235236&.
[10] M. Jones, “What Really Happened on Mars”, http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/ .
[11] Report to Congressional Committees. “DOD´s Goals for Resolving Space Based Infrared System Software Problems Are Ambitious.” General Accounting Office. GAO-08-1073. September 2008.
[12] Daniel L. Dvorak. NASA Study on Flight Software Complexity. Technical Report NASA/CR-2005-213912, NASA Office of Chief Engineer Technical Excellence Program, March 2009.
[13] Nancy Leveson. System Safety Engineering: Back To The Future. Technical Report MIT, 2002.
[14] Daniel Jackson Ed. “Software for Dependable Systems: Sufficient Evidence?” Committee on Certifiably Dependable Software Systems, National Research Council. National Academic Press, ISBN: 0-309-10857-8, 2007.
[15] VHSIC (Very High Speed Integrated Circuits) hardware description language en.wikipedia.org/wiki/VHDL.
[16] Model checking hardware en.wikipedia.org/wiki/Model_checking.
[17] Maurice Heitz, Sebastien Gabel, Julien Honoré, Xavier Dumas/CS, Iulan Ober/IRIT and David Lesens/ASTRIUM-ST Supporting a Multi-formalism Model Driven Development Process with Model Transformation, a TOPCASED implementation, Proceedings of 4th International Congress on Embedded Real-Time Systems, ERTS 2008.
[18] Eric Conquet “ASSERT: a step towards reliable and scientific system and software engineering” Proceedings of 4th International Congress on Embedded Real-Time Systems, ERTS 2008.
[19] SPICES: *Support for Predictable Integration of mission Critical Embedded Systems* http://www.spices-itea.org/
[20] STOOD: a real time software toolset supporting a range of methods and languages including UML 2, HRT HOOD, AADL, C, C++, and Ada 95.. www.ellidiss.com.
[21] P. Feiler, J. Hansson, D. de Niz, L. Wrage, “System Architecture Virtual Integration: An Industrial Case Study”, SEI Technical Report, CMU/SEI-2009-TR-017
November 2009.