

Report from Dagstuhl Seminar 12272

# Architecture-Driven Semantic Analysis of Embedded Systems

Edited by

Peter Feiler<sup>1</sup>, Jérôme Hugues<sup>2</sup>, and Oleg Sokolsky<sup>3</sup>

1 Carnegie Mellon University – Pittsburgh, US, [phf@sei.cmu.edu](mailto:phf@sei.cmu.edu)

2 ISAE – Toulouse, FR, [jerome.hugues@isae.fr](mailto:jerome.hugues@isae.fr)

3 University of Pennsylvania, Philadelphia, US, [sokolsky@cis.upenn.edu](mailto:sokolsky@cis.upenn.edu)

---

## Abstract

Architectural modeling of complex embedded systems is gaining prominence in recent years, both in academia and in industry. An architectural model represents components in a distributed system as boxes with well-defined interfaces, connections between ports on component interfaces, and specifies component properties that can be used in analytical reasoning about the model. Models are hierarchically organized, so that each box can contain another system inside, with its own set of boxes and connections between them.

The goal of Dagstuhl Seminar 12272 “Architecture-Driven Semantic Analysis of Embedded Systems” is to bring together researchers who are interested in defining precise semantics of an architecture description language and using it for building tools that generate analytical models from architectural ones, as well as generate code and configuration scripts for the system.

This report documents the program and the outcomes of the presentations and working groups held during the seminar.

**Seminar** 01.–06. July, 2012 – [www.dagstuhl.de/12272](http://www.dagstuhl.de/12272)

**1998 ACM Subject Classification** D.2.2 Design Tools and Techniques, D.2.4 Software/Program Verification, D.2.11 Software Architectures

**Keywords and phrases** Architectu Description Language, AADL, EAST-ADL, MARTE, Verification, Validation, Analysis, Embedded Systems, Model-Driven techniques

**Digital Object Identifier** 10.4230/DagRep.2.7.30

## 1 Executive Summary

*Peter Feiler*

*Jérôme Hugues*

*Oleg Sokolsky*

**License**  Creative Commons BY-NC-ND 3.0 Unported license  
© Peter Feiler, Jérôme Hugues, and Oleg Sokolsky

Architectural modeling of complex embedded systems is gaining prominence in recent years, both in academia and in industry. An architectural model represents components in a distributed system as boxes with well-defined interfaces, connections between ports on component interfaces, and specifies component properties that can be used in analytical reasoning about the model. Models are hierarchically organized, so that each box can contain another system inside, with its own set of boxes and connections between them. An architecture description language for embedded systems, for which timing and resource availability form an important part of the requirements, must describe resources of the system



Except where otherwise noted, content of this report is licensed under a Creative Commons BY-NC-ND 3.0 Unported license

Architecture-Driven Semantic Analysis of Embedded Systems, *Dagstuhl Reports*, Vol. 2, Issue 7, pp. 30–55  
Editors: Peter Feiler, Jérôme Hugues, and Oleg Sokolsky



Dagstuhl Reports  
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

platform, such as processors, memories, communication links, etc. Several architectural modeling languages for embedded systems have emerged in recent years, including AADL, SysML, EAST-ADL, and the MARTE profile for UML.

In the context of model-based engineering (MBE) architectural modeling serves several important purposes:

An architectural model allows us to break the system into manageable parts and establish clear interfaces between these parts. In this way, we can manage complexity of the system by hiding the details that are unimportant at a given level of consideration; Clear interfaces between the components allow us to avoid integration problems at the implementation phase. Connections between components, which specify how components affect each other, help propagate the effects of change in one component to the affected components. Most importantly, an architectural model can be seen as a repository of the knowledge about the system, represented as requirements, design, and implementation artifacts, held together by the architecture. Such a repository enables automatic generation of analytical models for different aspects of the system, such as timing, reliability, security, performance, etc. Since all the models are generated from the same source, ensuring consistency of assumptions and abstractions used in different analyses becomes easier. The first two uses of architectural modeling have been studied in the research literature for a number of years. However, the coordination role of architectural modeling in MBE is just currently emerging. We expect this role to gain importance in the coming years. It is clear that realizing this vision of "single-source" MBE with an architectural model at its core is impossible without having first a clear semantics of the architecture description language.

The goal of the seminar is to bring together researchers who are interested in defining precise semantics of an architecture description language and using it for building tools that generate analytical models from architectural ones, as well as generate code and configuration scripts for the system. Despite recent research activity in this area to use semantic interpretation of architectural models for analytical model generation, we observe a significant gap between current state of the art and the practical need to handle complex models. In practice, most approaches cover a limited subset of the language and target a small number of modeling patterns. A more general approach would most likely require an interpretation of the semantics of the language by the tool, instead of hard-coding of the semantics and patterns into the model generator.

## 2 Table of Contents

### Executive Summary

<i>Peter Feiler, Jérôme Hugues, and Oleg Sokolsky</i> . . . . .	30
---	----

### Overview of Talks


EAST-ADL – An Architecture-centric Approach to the Design, Analysis, Verification and Validation of Complex Embedded Systems <i>De-Jiu Chen</i> . . . . .	34
Model-Checking Support for AADL <i>Silvano Dal Zilio</i> . . . . .	34
Model-Based/ Platform-Based/Architecture-Driven Design of Cyber-Physical Systems <i>Patricia Derler</i> . . . . .	35
On the mechanization of AADL subsets <i>Mamoun Filali-Amine</i> . . . . .	35
Extended Literate Programming. Introducing the $\square$ (SquareCup) Language <i>Laurent Fournier</i> . . . . .	36
Software Component Architecture Model Analysis and Executable Generation using Semantic Language Layering <i>Serban Gheorghe</i> . . . . .	36
Architecture Evaluation @ Run-time: Problems, Challenges and Solutions <i>Lars Grunske</i> . . . . .	37
Embedded System Architecture for Software Health Management <i>Gabor Karsai</i> . . . . .	37
Experience of Using Architecture Models in Civil Aviation Domain <i>Alexey Khoroshilov</i> . . . . .	38
Hierarchy is Good For Discrete Time: a Compositional Approach to Discrete Time Verification <i>Fabrice Kordon</i> . . . . .	38
Formal Semantics of AADL Component Behavior to Prove Conformance to Specification <i>Brian Larson</i> . . . . .	39
Software Architecture Modeling by Reuse, Composition and Customization <i>Ivano Malvolta</i> . . . . .	40
Architecture-Driven analysis with MARTE/CCSL <i>Frédéric Mallet</i> . . . . .	41
Approximating physics in the design of technical systems <i>Pieter J. Mosterman</i> . . . . .	41
Satellite Platform Case Study with SLIM and COMPASS <i>Viet Yen Nguyen</i> . . . . .	41

Correctness, Safety and Fault Tolerance in Aerospace Systems: The ESA COMPASS Project <i>Thomas Noll</i> . . . . .	42
Synchronous AADL: From Single-Rate to Multirate <i>Peter Csaba Ólveczky</i> . . . . .	42
Semantic anchoring of industrial architectural description languages <i>Paul Petterson</i> . . . . .	43
Integration of AADL models into the TTEthernet toolchain; Towards a model-driven analysis of TTEthernet networks <i>Ramon Serna Oliver</i> . . . . .	44
Architecture Modeling and Analysis for Automotive Control System Development <i>Shin'ichi Shirashi</i> . . . . .	44
About architecture description languages and scheduling analysis <i>Frank Singhoff</i> . . . . .	44
Co-modeling, simulation and validation of embedded software architectures using Polychrony <i>Jean-Pierre Talpin</i> . . . . .	45
Compositional Analysis of Architecture models <i>Michael W. Whalen</i> . . . . .	45
<b>Working Groups</b>	
Attaching semantics to a modeling framework . . . . .	46
Expressive Power of Architectural Models . . . . .	47
Analysis of architectural systems . . . . .	48
Multi-point view analysis and combination of analysis result . . . . .	49
Run-time Architectural Analysis . . . . .	49
Notion of Time: Physical vs. Real-Time vs Discrete vs Logical time . . . . .	50
Patterns for (de)composing and analysing systems . . . . .	51
<b>Summary and Open Challenges</b> . . . . .	52
<b>Bibliography</b> . . . . .	53
<b>Participants</b> . . . . .	55

### 3 Overview of Talks

#### 3.1 EAST-ADL – An Architecture-centric Approach to the Design, Analysis, Verification and Validation of Complex Embedded Systems

*De-Jiu Chen (KTH – Stockholm, SE)*


License  Creative Commons BY-NC-ND 3.0 Unported license  
© De-Jiu Chen

EAST-ADL is a domain specific Architecture Description Language (ADL) for safety-critical and software-intensive embedded systems. The language enables a formalized and traceable description of a wide range of engineering concerns throughout the entire lifecycle of systems. This makes it possible to fully utilize the leverage of state-of-the-art methods and tools for the development of correct-by-construction system functions and components in a seamless and cost efficient way.

This talk focuses on the recent advances of EAST-ADL in supporting the description, analysis, verification&validation of complex embedded systems for the purposes of requirements engineering, application design, and safety engineering. The approach is architecture centric as all behavior descriptions are formalized and connected to a set of standardized design artifacts existing at multiple levels of abstraction. This talk presents the language design, its theoretical underpinning and tool implementation. From a bigger perspective, the contribution makes it possible for embedded system and software developers to maintain various engineering concerns coherently, while exploiting mature state-of-the-art technologies from computer science and other related domains for a model-based design.

#### 3.2 Model-Checking Support for AADL

*Silvano Dal Zilio (LAAS – Toulouse, FR)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Silvano Dal Zilio


We present recent work on the extension of the Fiacre language with real-time constructs and real-time verification patterns. We will show how these enhancements have been used to implement a new version of a model-checking toolchain for AADL.

Fiacre is a formal language with support for expressing concurrency and timing constraints; its goal is to act as an intermediate format for the formal verification of high-level modeling language, such as UML profiles for system modeling. Essentially, Fiacre is designed both as the target of model transformation engines, as well as the source language of compilers into verification toolboxes, namely Tina and CADP.

Our motivations for extending Fiacre are to reduce the semantic gap between Fiacre and high-level description languages and to streamline our verification process. We will take the example of a transformation from AADL (and its behavioral annex) to Fiacre to explain the benefit of this new toolchain

### 3.3 Model-Based/ Platform-Based/Architecture-Driven Design of Cyber-Physical Systems

*Patricia Derler (University of California – Berkeley, US)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Patricia Derler

This presentation focuses on recent efforts of including architectural properties into executable models in Ptolemy II. A programming model that includes some architecture information such as Sensors, Actuators, Platforms, Execution Time, Memory is Ptides, which is also implemented in Ptolemy. The talk describes the Ptides execution semantics and its implementation in Ptolemy. Evaluation of architectural properties and constraints is done via simulation.

### 3.4 On the mechanization of AADL subsets

*Mamoun Filali-Amine (Paul Sabatier University – Toulouse, FR)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Mamoun Filali-Amine

AADL (Architecture Analysis & Design Language) is a real-time specification language that focuses on the early analysis of the dynamic architecture of a system and the correctness of resource allocation described by the software / hardware mapping. Although, these aspects are precisely described in the standard, they need a formal expression in order to be able to verify their properties formally.

With respect to formal verification, it is interesting to consider two kind of properties:

- applicative properties like schedulability, absence of buffer overflows, deadlocks, starvation and more generally safety and liveness properties. The first ones are application independent while the last ones depend on the intended behavior of the application.
- properties related to the semantics of the language constructs, e.g., determinism, correctness of some model transformations (flattening, application of distribution strategies, ...).


While for applicative properties, model checking techniques have been widely and successfully applied, proof based techniques are still necessary to address properties which in general do not concern a fixed finite domain.

In this talk, we will present the different proof based attempts that we have conducted in order to mechanize some aspects of AADL.

- Semantics of basic AADL protocols related to threads and communication.
- Semantics of a basic AADL computation model.
- Mechanization of the basic semantics of the FIACRE language and proof support for a parameterized version.

### 3.5 Extended Literate Programming. Introducing the $\sqcup$ (SquareCup) Language

Laurent Fournier (Rockwell Collins France, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Laurent Fournier


We introduce a universal typed sparse graph language called " $\sqcup$ " (the *square cup* character, Unicode #2294, or `\sqcup` in `TeX`).  $\sqcup$  extends the established *Literate Programming* (LP) approach with Model Driven Engineering (MDE) paradigm. Unlike *Literate Modeling* that basically produces documented diagrams reports, our *Extended Literate Programming* (ELP) proposal focus on satisfying the two original LP goals; full **automation** from the highest level declarative description to build a runnable machine code, and better **understanding** to produce high quality shared documentation, all in a literate form and within the same capture tool.

$\sqcup$  graph nodes support the useful notion of *port* and the  $\sqcup$  simple text syntax – concision and readability to compare for instance with XMI serialization format – can represent any kind of nested graphs. Supported formalisms vary but not limited to UML, *SysML*, *AADL graph*, *Simulink/Scicos/Scade blocs*, *State machine*, *Markov chain*, *Petri Nets*, *EntityRelation graph*, KAOS. . . For rendering, diagram positioning/routing attributes are excluded to rely only on automatic layout algorithms. All the semantics of a  $\sqcup$  graph is defined in nodes and arc *types* by a mapping to a particular generated code construction. Those *types* allow to build some DSL and Domain Specific Libraries of components for easy *Product Line Development*. ELP facilitates Requirement Engineering tasks like traceability, impact analysis, and transformations to design phase. Because each  $\sqcup$ -*tool* instance natively provides a  $\sqcup$  models library and a *types* library available as a remote service, the designer can access to a set of cooperating/competing code generators, building a simple *Semantic Web* for software engineering. For any  $\sqcup$  graph, the same techniques apply to generate SVG code for web browsing, *TikZ* code for document embedded diagrams, than for generating compilable code. Furthermore, the node/arc content is a free *Unicode* string parsed as a *Python* interpreter so all downstream transformations works on the *Python Abstract Syntax Tree* (PAST). A tool framework is under development, using a web based editor (*CodeMirror*), optimized PDF rendering, a *Git* database on the cloud and *Python3* as glue language. Code generation will focus on AADL interpretation first. Unlike WYSIWYG (*What You See Is What you Get*) frameworks, the text nature of  $\sqcup$  models makes save, search, diff or merge operations easy and long term resistant.

The  $\sqcup$  project is currently hosted at <https://github.com/pelinquin/u>.

### 3.6 Software Component Architecture Model Analysis and Executable Generation using Semantic Language Layering

Serban Gheorhe (Edgewater Computer Systems Inc. – Ottawa, CA)

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Serban Gheorghe


Without executable code generation, models of software systems are inevitably relegated to the role of exploration of design alternatives and design documentation. They will inevitably represent an alternative and potentially stale “source of truth” of the real running software,

usually implemented in programming languages accepted in the industry. In this talk, we focus on executable software applications described using the AADL software component model and compliant with the AADL run-time execution semantics. The AADL run time semantics accommodates multiple possible computation model choices (synchronous/asynchronous, preemptable/non-preemptable, etc.). Also, target execution platforms currently in use have a wide variety of possible execution semantics. It would be extremely costly to build trusted code generators for all possible combinations affecting the AADL run-time execution semantics that also preserve at run-time the formal temporal properties expressed and proven by static analysis on the AADL model. Our approach is to select a small anchor subset of the AADL run-time semantics, called the RTEdge AADL Microkernel subset, and use it as a trusted lower level semantic layer, encoded in a run-time executive middleware (called RTExec) available as a library on multiple execution platforms. Corresponding to the RTExec semantics we define an AADL language subset, called the RTEdge modeling language subset, which is executable via code generation and linking with the RTExec executive.

Given an AADL software system, we use this semantic layering to generate software executables for any execution platform that runs the RTExec: firstly by translating the AADL source into the intermediate RTEdge subset, thus obtaining an RTEdge model with equivalent execution semantics, secondly by generating target specific executables from the RTEdge model. Formal temporal properties expressed at the AADL run-time semantic layer can be mapped into equivalent temporal properties expressed and checked at the lower and fix semantic layer of the RTEdge subset, guaranteeing consistency between the analysis assumptions and the real execution semantics of the generated executable.

### 3.7 Architecture Evaluation @ Run-time: Problems, Challenges and Solutions


*Lars Grunske (TU Kaiserslautern, DE)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Lars Grunske

The majority of innovations in modern technical systems are driven by software. Based on the research results of the software engineering community over the past decades we are able to develop software systems with immense complexity. However, concomitant with these increases in complexity, the quality demands also appear to be ever-growing. Probabilistic properties defined in probabilistic temporal logics are commonly applied to specify these quality demands and are especially suitable for performance, reliability, safety, and availability requirements. This talk will present approaches but also problems and challenges for run-time architecture evaluation strategies of these probabilistic properties.

### 3.8 Embedded System Architecture for Software Health Management

*Gabor Karsai (Vanderbilt University, US)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Gabor Karsai


As software increasingly becomes the main source of functionality and the ultimate tool for system integration in cyber-physical systems there is an increasing chance that imperfections



in the design and implementation of the software need to be detected and mitigated at run-time. Such needs can be addressed by borrowing metaphors and techniques from the area of 'systems health management', where the concepts and technologies of anomaly detection, fault source isolation, and fault mitigation have been developed. Similarly to physical systems, a software health management (SHM) approach necessitates an architectural foundation: a component framework that defines units for fault management and fault containment, with precisely specified and controlled interfaces and interactions. Based on this foundation a highly reusable software health management layer can be constructed that maintains system functionality even when software defects appear. This layer is model-based: it consists of generic building blocks and algorithms that are configured via models. Using an architectural framework, with precisely defined component interaction semantics enables not only the implementation but the formal verification and analysis of the entire system. The talk introduces a motivating example, presents a component framework and how it was extended to support software health management, and concludes with a realistic case study.

### 3.9 Experience of Using Architecture Models in Civil Aviation Domain


*Alexey Khoroshilov (Russian Academy of Sciences – Moscow, RU)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Alexey Khoroshilov

The talk presents an experience of building an AADL-based toolset for integrated modular avionics (IMA) design and integration. Features and an architecture of the toolset are described and the role of the architecture description language is discussed.

### 3.10 Hierarchy is Good For Discrete Time: a Compositional Approach to Discrete Time Verification

*Fabrice Kordon (UPMC – Paris, FR)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Fabrice Kordon

#### Introduction

Model checking is now widely used as an automatic and exhaustive way to verify complex systems. However, this approach suffers from an intrinsic combinatorial explosion, due to both a high number of synchronized components and a high level of expressivity in these components.

With respect to the expressivity issue, we consider the particular problem of introducing explicit time constraints in the components of a system. In this modeling step, the choice of a time domain is important, impacting on the size of the resulting model, the class of properties which can be verified and the performances of the verification.

In this presentation, we show that hierarchical encoding of elementary components encapsulating labeled transition systems (LTS), synchronized by means of public transitions, is an efficient way to encode discrete time.

## Instantiable Transition Systems

Instantiable Transition Systems (ITS) are a framework designed to exploit the hierarchical characteristics of SDD [5]. This structure is used to encode the state space, for the description of component based systems. ITS were introduced in [11]. Here are the main principles ITS rely on:

- ITS types (elementary) represent a LTS and export some public transitions that can be synchronized with other ITS types,
- composite ITS gather several ITS (composite or elementary) and propose a new interface that can be connected to some of the synchronized public actions of enclosed ITS,
- instantiation allows to create a number of entities having the same behavior. This emphasizes the description of regularities in distributed systems.

## Encoding discrete time with ITS

The basic idea of using ITS to model discrete time is to propose an extra interface dedicated to time elapse [10]. This interface interacts with local clocks. When time elapse the same way all over the system, the elapse interfaces must be synchronized together. It is also possible to have local synchronization of clocks to model several timelines.

This presentation shows the main principles of this mechanism and illustrates it on a simple example from the literature. Then, we emphasize its interest in a small medical case study: the Body Area Network. Here, ITS are generated from a high-level language dedicated to the description of Wireless Sensor Networks: Verisensor [4].

### 3.11 Formal Semantics of AADL Component Behavior to Prove Conformance to Specification


*Brian Larson (Multitude Corp., US)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Brian Larson

Software can be more dependable than hardware. This requires programs, their specifications, and their executions are mathematical objects, so that conformance to specification of every execution can be formally verified. This talk presents an AADL annex sublanguage, Behavioral Languages for Embedded Systems with Software (BLESS) to formally define component behavior, based on the Behavior Annex (BA) language. BLESS adds Assertions to form proof outlines that can be transformed into complete formal proofs semi-automatically.

### 3.12 Software Architecture Modeling by Reuse, Composition and Customization

*Ivano Malavolta (Univ. degli Studi di L'Aquila, IT)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Ivano Malavolta

While developing a complex system, it is of paramount importance to correctly and clearly specify its software architecture. Architecture Description Languages (ADLs) are the means to define the software architecture of a system. ADLs are strongly related to stakeholder concerns: they must capture all design decisions fundamental for systems stakeholders. From the earliest work in software architecture, the usefulness of expressing software architectures in terms of multiple views is well recognized. Architecture views represent distinct aspects of the system of interest and are governed by viewpoints which define the conventions for their construction, interpretation and use to frame specific system concerns. Most practicing software architects operate within an architecture framework which is a coordinated set of viewpoints, models and notations prescribed for them. As a matter of fact, stakeholder concerns vary tremendously, depending on the project nature, on the domain of the system to be realized, etc. So, even if current architecture frameworks are defined to varying degrees of rigor and offer varying levels of tool support, finding the right architecture framework that allows to address the various system concerns is both a risky and difficult activity. Therefore, an effective way to define and combine architectural elements into a suitable framework for effectively create architecture descriptions is still missing.


In this presentation, I propose an infrastructure for modeling the architecture of a software system by adapting existing architectural languages, viewpoints and frameworks to domain- and organization-specific features. Under this perspective, the proposed infrastructure allows architects to set up customized architectural frameworks by: (i) defining and choosing a set of viewpoints that adequately fit with the domain and features of the system being developed, (ii) automatically adapting existing architecture description languages to project-specific concerns, (iii) keeping architectural views within the framework synchronized, (iv) enabling consistency and completeness checks based on defined correspondences and rules among architectural elements. The proposed approach builds upon the conceptual foundations of ISO/IEC/IEEE 42010 for architecture description and it is generic with respect to the used architectural elements (i.e., views, viewpoints, languages, stakeholder's concerns, etc.).

The impact of the proposed approach is three-fold: (i) a novel approach is presented for architecting by reusing, composing and customizing existent architectural elements, (ii) a new composition mechanism is presented for extending architectural languages in a controlled fashion, (iii) a new mechanism for keeping architectural views in a consistent state is provided.

The proposed approach is realized through a combination of model transformations, weaving, and megamodeling techniques. The approach has been put in practice in different scenarios and has been evaluated in the context of a real complex system.

### 3.13 Architecture-Driven analysis with MARTE/CCSL

*Frédéric Mallet (INRIA Sophia Antipolis, FR)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Frédéric Mallet


The UML Profile for Modeling and Analysis of Real-Time and Embedded systems promises a general modeling framework to design and analyze systems. Lots of works have been published on the modeling capabilities offered by MARTE, much less on verification techniques supported. The Clock Constraint Specification Language (CCSL), first introduced as a companion language for MARTE, was devised to offer a formal support to conduct causal and temporal analysis on MARTE models.

This presentation focuses on the analysis capabilities of MARTE/CCSL and describes a process where the logical description of the application is progressively refined to take into account the execution platforms (software and hardware architectures) and the environment constraints.

The approach is illustrated on two very simple examples where the architecture plays an important role. During the presentation, issues are raised on the expressiveness of CCSL, on the nature of properties that can be analyzed and on possible extensions.

### 3.14 Approximating physics in the design of technical systems


*Pieter J. Mosterman (The MathWorks Inc. – Natick, US)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Pieter J. Mosterman

In the design of Cyber-Physical Systems, physics plays a crucial role. Models of physics at a macroscopic level often comprise differential and algebraic equations. These equations typically require computational approaches to derive solutions. Approximations introduced by the solvers that derive these solutions to a large extent determine the meaning of the models, in particular when discontinuities are included. In reasoning about models that are solved computationally it is therefore imperative to also model the solvers. This presentation shows how performance of a cyber-physical system may be affected by physics and conceptualizes the modeling of computational solvers. Opportunities that derive from the availability of solver models are presented and a control synthesis approach for stiff hybrid dynamic systems based on model checking is outlined.

### 3.15 Satellite Platform Case Study with SLIM and COMPASS

*Viet Yen Nguyen (RWTH Aachen, DE)*


License  Creative Commons BY-NC-ND 3.0 Unported license  
© Viet Yen Nguyen

This talk is a continuation of Thomas Noll's talk on SLIM, a formalized dialect of AADL. We report on the use of the COMPASS toolset on a satellite platform in development at the European Space Agency. These efforts were carried out in parallel with the conventional software development of the satellite. The nominal behavior of the satellite platform model,

expressed in SLIM, comprises nearly 50 million states. This multiplies manifold upon the injection of failures. We show that verification and validation artifacts, typically constructed manually in a space system engineering process, can be automatically generated by the COMPASS toolset. The model's size pushed the computational tractability of the algorithms underlying the formal analyses, and revealed bottlenecks for future theoretical research. Additionally, the effort led to newly learned practices from which subsequent formal modeling and analysis efforts shall benefit, especially when they are injected in the conventional software development lifecycle. The case demonstrates the feasibility of fully capturing a system-level design as a single comprehensive formal model and analyze it automatically using a formal methods toolset based on (probabilistic) model checkers.

### 3.16 Correctness, Safety and Fault Tolerance in Aerospace Systems: The ESA COMPASS Project

*Thomas Noll (RWTH Aachen, DE)*


License  Creative Commons BY-NC-ND 3.0 Unported license  
© Thomas Noll

Building modern aerospace systems is highly demanding. They should be extremely dependable, offering service without failures for a very long time – typically years or decades. The need for an integrated system- software co-engineering framework to support the design of such systems is therefore pressing. However, current tools and formalisms tend to be tailored to specific analysis techniques and do not sufficiently cover the full spectrum of required system aspects such as safety, dependability and performability. Additionally, they cannot properly handle the intertwining of hardware and software operation. As such, current engineering practice lacks integration and coherence.

This talk gives an overview of the COMPASS project that was initiated by the European Space Agency to overcome this problem. It supports system- software co-engineering of real-time embedded systems by following a coherent and multidisciplinary approach. We show how such systems and their possible failures can be modeled in (a variant of) AADL, how their behavior can be formalized, and how to analyze them by means of model checking and related techniques. Practical experiences obtained in a larger case study will be described in a subsequent presentation by Viet Yen Nguyen.

### 3.17 Synchronous AADL: From Single-Rate to Multirate

*Peter Csaba Ölveczky (University of Illinois – Urbana, US)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Peter Csaba Ölveczky

Distributed Real-Time Systems (DRTSs), such as avionics systems and distributed control systems in motor vehicles, are very hard to design because of asynchronous communication, network delays, and clock skews. Furthermore, their model checking typically becomes unfeasible in practice due to the large state spaces caused by the interleavings. Based on the observation that many automotive and avionics systems should be *virtually synchronous*—that is, conceptually, there is a logical period during which all components perform a transition and send messages to each other—that must be realized in a distributed environment with

network delays, skewed local clocks, etc., we have proposed the PALS transformation [8, 9]. The key idea of PALS (“physically asynchronous logically synchronous”) is that one can model and verify the much simpler synchronous design, and PALS then provides a correct-by-construction distributed asynchronous model.





To make the PALS modeling and verification methodology available to the modeler, we have defined an annotated sublanguage of AADL, called *Synchronous AADL*, that can be used to specify synchronous PALS designs in AADL [2]. We have defined the formal semantics of Synchronous AADL in Real-Time Maude, and have used this semantics to develop an OSATE plug-in, called *SynchAADL2Maude*, that provides simulation and temporal logic model checking for synchronous designs modeled in Synchronous AADL *within* OSATE [3]. This enables a model-engineering process for important classes of distributed real-time systems that combines the convenience of AADL modeling, the complexity reduction of PALS, and formal verification in Real-Time Maude. We have used *SynchAADL2Maude* on a virtually synchronous avionics system whose distributed asynchronous version (even in very simple settings) has millions of reachable states and cannot be feasibly model checked, but where the Synchronous AADL model of the corresponding synchronous PALS design could be verified by the *SynchAADL2Maude* tool in less than a second.

However, a number of DRTSs are *multirate* systems whose components have different periods. For example, the controller for the ailerons on an airplane may operate with a period of 15 ms, whereas the rudder controller operates with period 20 ms. These different components need to synchronize when turning the airplane. We have therefore extended PALS to multirate virtually synchronous systems, and are working on extending the *SynchAADL2Maude* tool to specify and verify such systems. That work could be based on the support for modeling multirate systems in AADL that has recently been developed by colleagues at UIUC and Rockwell Collins [1].

This presentation is based on joint work with José Meseguer, Kyungmin Bae, Lui Sha, Abdullah Al-Nayem, Steven P. Miller, and Darren D. Cofer.

### 3.18 Semantic anchoring of industrial architectural description languages


*Paul Pettersson (Mälardalen University – Västerås, SE)*

License     Creative Commons BY-NC-ND 3.0 Unported license  
© Paul Pettersson

In some recent work, we have focused on defining semantics to industrial architecture description languages, such as Procom, AADL and EAST-ADL. Semantic anchoring of such languages has served different purposes. A main goal has been to enable analysis of the languages in analysis tools such as simulators and model-checkers, including UPPAAL and UPPAAL PORT. Current work is focusing on addressing dynamically reconfiguring systems and model-based testing using ADLs.

### 3.19 Integration of AADL models into the TTEthernet toolchain; Towards a model-driven analysis of TTEthernet networks

*Ramon Serna Oliver (TTTech Computertechnik – Wien, AT)*


License  Creative Commons BY-NC-ND 3.0 Unported license  
© Ramon Serna Oliver

As networked cyber-physical embedded systems become more and more populated, models to enable semantic analysis are a key factor to reduce complexity. We introduce Time-Triggered Ethernet (TTEthernet) and the TTE Tool-chain and explore the advantages of a complete system representation by means of the Architecture Analysis and Design Language (AADL).

The presentation elaborates on the use of AADL at different levels. Namely: providing a manageable representation of a (potentially complex) network; introducing a comprehensible interface to and within TTE-Tools; building a repository of system components, properties, constraints, and requirements; a means to network analysis and property verification; and, an open door to complex analysis (through existing tools, annexes, etc...).

### 3.20 Architecture Modeling and Analysis for Automotive Control System Development

*Shin'ichi Shirashi (TOYOTA InfoTechnology Center USA Inc., US)*


License  Creative Commons BY-NC-ND 3.0 Unported license  
© Shin'ichi Shirashi

Architecture modeling languages, e.g., AADL, SysML, and MARTE are well known languages and used among several different domains. This talk explains modeling steps based on these languages through a real-world automotive system example. On the other hand, this talk also explains our experience of architecture analysis from the real-time automotive system viewpoint.

In the end, the relation and gap between the architecture model and architecture analysis are discussed.

### 3.21 About architecture description languages and scheduling analysis

*Frank Singhoff (University of Brest, FR)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Frank Singhoff


The talk deals with performance verifications of architecture models. We focus on real-time embedded systems and their verification with the real-time scheduling theory.

Many industrial projects do not perform performance analysis with this theory even if the demand for the use of it is large. To perform verifications with the real-time scheduling theory, the architecture designers must check that their models are compliant with the assumptions of this theory. Unfortunately, this task is difficult since it requires that designers have a deep understanding of the real-time scheduling theory. In this presentation, we show how to help designers to check that an architecture model is compliant with the real-time scheduling theory assumptions.

We focus on schedulability tests. We show how to explicitly model the relationships between an AADL architectural model and schedulability tests. From these models, we apply a model-based engineering process to generate tools which are able to check compliance of architecture models with schedulability tests assumptions.

### 3.22 Co-modeling, simulation and validation of embedded software architectures using Polychrony

*Jean-Pierre Talpin (INRIA – Rennes, FR)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Jean-Pierre Talpin

The design of embedded systems from multiple views and heterogeneous models is ubiquitous in avionics as, in particular, different high-level modeling standards are adopted for specifying the structure, hardware and software components of a system. The system-level simulation of such composite models is necessary but difficult task, allowing to validate global design choices as early as possible in the system design flow. This paper presents an approach to the issue of composing, integrating and simulating heterogeneous models in a system co-design flow. First, the functional behavior of an application is modeled with synchronous data-flow and statechart diagrams using Simulink/Genesys. The system architecture is modeled in the AADL standard. These high-level, synchronous and asynchronous, models are then translated into a common model, based on a polychronous model of computation, allowing for a Globally Asynchronous Locally Synchronous (GALS) interpretation of the composed models. This compositional translation is implemented as an automatic model transformation within Polychrony, a toolkit for embedded systems design supporting simulation, verification, controller synthesis, sequential and distributed code-generation. An avionic case study, consisting of a simplified doors and slides control system, is presented to illustrate our approach.

### 3.23 Compositional Analysis of Architecture models

*Michael W. Whalen (University of Minnesota, US)*

License  Creative Commons BY-NC-ND 3.0 Unported license  
© Michael W. Whalen

This presentation describes work towards a design flow and supporting tools to improve design and verification of complex cyber-physical systems. We focus on system architecture models composed from libraries of components and complexity-reducing design patterns having formally verified properties. This allows new system designs to be developed rapidly using patterns that have been shown to reduce unnecessary complexity and coupling between components. Components and patterns are annotated with formal contracts describing their guaranteed behaviors and the contextual assumptions that must be satisfied for their correct operation. We describe the compositional reasoning framework that we have developed for proving the correctness of a system design, and provide a proof of the soundness of our compositional reasoning approach. An example based on an aircraft flight control system is provided to illustrate the method and supporting analysis tools.



## 4 Working Groups

During the seminar, the group discussed possible topics for break-out sessions, and form different working groups. We placed two afternoon sessions dedicated to brainstorming session on Tuesday and Thursday afternoon.

The topics were the following, on Tuesday afternoon:

- Attaching semantics to a modeling framework (section 4.1)
- How much expressive power is needed for architectural models (section 4.2)
- Analysis of architectural systems (section 4.3)

on Thursday afternoon:

- Multi-point view analysis and combination of analysis result (section 4.4)
- Run-time Architectural Analysis (section 4.5)
- Notion of Time: Physical vs. Real-Time vs Discrete vs Logical (section 4.6)
- Patterns for (de)composing and analysing systems (section 4.7).

### 4.1 Attaching semantics to a modeling framework

**Motivation.** The group acknowledged the fact that semantics is a key enabler to perform further analysis. Actually, most analysis require hypothesis on the behavior of the system, the interconnection between elements to follow some particular semantics: typing system, execution semantics, propagation of information/events.

Also, the group recognized that this area is usually not well developed in many tools: semantics is usually part of the analysis tool itself, except for some consistency checking performed during model analysis and transformation. Actually, the interpretation of semantics as done by a tool is usually not explicit and remains hidden.

The group contemplated different options:

- **at tool-level:** this creates a strong link to a particular tool, and can be perceived as a vendor lock-in strategy.
- **at model-level:** the model can not only store user artifacts, but also the underlying semantics. This could enable a wider sharing and understanding of how to interpret a model.

Furthermore, analysis tools can use this additional information. Yet this poses the question of the mechanisms to encode this semantics.

- **as part of the transformation process:** yet, this still creates a strong link between the tools, mostly the editing and processing parts, e.g. ECLIPSE.

**Open questions and discussions.** From the different options raised, the key question is “*What is expected from the tool?*”. There might be several expectations from the users (design teams, project management, tool builder, ...).

Building consistent semantics is a complex work, usually performed using operational semantics or equivalent frameworks. Attaching semantics, even if it is a desirable effort may create an artifact that is, in the end, too-complex to be manipulated. A corollary to the previous question is “*Is formal too formal?*”.

A good compromise seems to opt for the following strategy: 1) define a DSL to define the model, with an implicit semantics easy to understand thanks to well-chosen concepts, 2) define separately the formal semantics, in a readable format but only for people interested in the core details. Such strategy would enable meeting certification requirements. Also,

defining separately the semantics would enable an adaptation on a per tool basis: scheduling and fault analysis could be defined based on two separate yet compatible description of the semantics of the system.

## 4.2 Expressive Power of Architectural Models

**Motivation.** A well-known trade-off in modeling is that, while increasing expressive power of a modeling language makes construction of models easier, it also makes analysis of models more difficult. Therefore, for an architectural modeling and analysis framework, it is important to identify the right level of expressiveness. Architectural analysis is intended to cover large-scale systems and systems of systems. This seems to imply that expressive power of architectural models and the level of detail in models cannot be too high. At the same time, an architectural modeling framework should not be viewed as a single modeling language, but rather as a collection of complementary languages that concentrate on different aspects of the system design, so that a model of a particular aspect can be simple and does not have to require much expressive power.

**Generative techniques in architectural analysis.** The key to an architecture-centric modeling framework is its reliance of generative techniques and model transformations. At the core of the framework, an architectural model serves as a repository of knowledge about the system. To perform analysis of a certain aspect of the system, an analysis model is generated from the architectural model that contains only the details needed for the selected analysis, and the expressive power of the language for the generated model is similarly targeted to, and limited by, the needs of the analysis. On the other hand, expressive power of the architectural description affects complexity of generation algorithms.

Generation of analysis models is enabled by the semantic core of the architectural framework that helps ensure that generated analysis models are consistent with each other and provides the basis for proving correctness of the generation. The semantic core is built on a semantic domain (e.g., sets of event sequences). Elements of the language establish relationships between elements of the domain. The logic for expressing these relationships can be general and expressive. We can use domain-specific languages to limit expressive power as needed.

An important aspect of a framework such as AADL is the ability to extend models through custom properties and annexes. Such an extension mechanism allows us to use different formalisms for different aspects of the system. The challenge is to use the extension mechanism in a way that keeps extensions compatible. Semantics of the extension mechanism itself become important here. In particular, when the extension is done by means of a new property set, a suitable semantics for properties may be by means of equations or constraints that relate values of properties in the set to each other as well as to concepts from the semantic core.

**Architectural vs. behavioral modeling.** Many architecture-level analysis techniques are concerned with operational aspects of the system. For example, timing and schedulability analysis relies on high-level thread execution semantics. Other analysis techniques can refine these high-level semantics with additional details of application behavior. There is much discussion in the community about the distinction between behavioral and architectural modeling and analysis, and whether the addition of behavioral details to an architectural model turns it into a behavioral model, defeating the promise of architecture-driven, system-level analysis. If this is indeed a valid concern, a question to be addressed by the research

community is to decide whether the modeling language should enforce the acceptable level of behavioral detail.

It has been pointed out that the distinction between behavior and architecture may lie in the kind of analysis that is applied and not in the formalism *per se*. For example, a system of differential equations can be simulated, which involves computing the flows in the system. This is a clear example of behavioral analysis. On the other hand, the same system of equations can be used for structural analysis, which is, essentially, an architecture-level technique.

### 4.3 Analysis of architectural systems

**Motivation.** The breakout session started with a presentation of an example by Gabor Karsai of a multi-mission versatile constellation of satellites: the Fractionated Space System Architecture. Each satellite carries different instruments. The combination of satellites allows for complex missions. There are multi-scale architectural issues in this system: satellite-level, constellation-level to dimension network, scheduling, fault management logic or energy. But also mission-level challenges, e.g. to select the best configuration, but also the satellites most suitable for a particular mission.

The group noted that the architecture of the system is central to the analysis. Given the complexity of the whole constellation, one needs to perform careful and clever separation of concerns along the various dimensions. One needs to distinguish:

- Architectural level at which a property can be assessed
- Property determination as part of lifecycle: permanent, instance-specific
- V&V techniques: model checking, test, proof, validation, runtime monitoring, etc
- Time available for V&V effort to be bounded

**Open questions and discussions.** From these considerations, we note there are several issues related to the analysis strategy to be deployed:

We first note that the analysis strategy, as part of the design or the V&V qualification effort, is costly. A first element of choice is therefore the running-time assumptions one may attach to particular activities (e.g. model checking vs. static analysis of a pattern) Composability of analysis across architectural layers could also reduce this effort.

A first solution would be to properly document analysis in the form of contract passed to architecture elements. Pre-conditions are expected patterns, properties, post-conditions are new elements deduced and propagated to the architecture. Although the group agrees on general list of V&V techniques and objectives, we acknowledged such document is lacking to the community.

Another issues reported by the group is a “chicken/egg” problem among analysis. Some analysis are cross-dependent, but more important is that the output of these analysis are of equal importance to the designer, for instance parameter configuration of the system. Therefore, one needs to apply some particular optimization or SAT problem solving strategies at the architectural level.

We note these two open questions pose a strong constraint on tooling support. We note that analysis tools are usually viewed as “extensions” plug-in to the modeling environment. Actually, we may need to view it the other way: the analysis framework is central, and considers the actual architecture descriptions as “plug-in” from which it extracts relevant information and support the designer activities.

#### 4.4 Multi-point view analysis and combination of analysis result

**Motivation.** Analyzing a complex embedded systems involve combining a model of the system under consideration (architecture, functional and non-functional properties) and analysis tools. We note several issues in this domain.

First, “properties” is a fuzzy term used to define either

1. what to describe: the way the system is from a set of simple classifiers, e.g. a task priority;
2. or what to assess: more complex elements deduced from the interacting components, answering an elaborate question “is the system schedulable? safe, etc”?

Besides, properties are either defined by the designer, or output of a particular analysis. A typical example being priorities applied to thread that can be either enforced or deduced from other elements. If we continue in the field of scheduling, it could be Rate Monotonic Analysis for deciding schedulability of a system based on actual priority value, or computing a priority assignment from Deadline Monotonic Analysis, etc.

Finally, we note, from the variety of Architecture Description Languages presented (MARTE, EAST-ADL, AADL, but also Simulink) that common properties may differ in names (WCET, Compute\_Execution\_Time), semantics (fixed value, range, ...), but also make assumption on units (“ticks” vs. actual time units). Finally, they can be used in different contexts, to represent an assumption, a budget, a computed, measured or refined value depending on the process and the maturity of the model.

**Open questions and discussions.** We note a strong interaction between models and analysis. Actually, the “final” system after the whole design effort is actually an iterative fix-point where properties and model are no longer evolving. This raises some questions on how to combine analysis in a way that eases convergence to this fix point.

The group note this is highly related to the modeling process in place, and that some elements of solutions already exist. For instance, EAST-ADL has been defined so that project manager knows elements to be modeled for each steps of its design, how to relate those steps to analysis, and how to build new properties at step N+1 from analysis performed at step N. Such process is highly specific to the automotive domain covered by EAST-ADL that constrains the system dimension, it is not available for generic ADL like MARTE or AADL.

Another open question is about the role of analytical model in the whole design process. In MARTE or AADL, this analytical model is a by-product, that is not kept. Yet, some analysis are time consuming (e.g. simulations, model checking), knowing whether they should be redone is a critical part to help converging to the final solution. An option is to determine when a model has evolved in a way that impact the analytical model.

The group discussed possible options to tackle this issue: one may consider an analytical model to be: an actual result (yes/no, numerical values, etc.) and a contract set on the model, defining invariants to be preserved so that the result is preserved, e.g. architectural patterns, properties to be maintained over time. If one of the invariant is broken, the analytical model should be rebuilt. Such concept, although appealing to the mind needs to be further refined and applied to the underlying meta-modeling framework, e.g. EMF.

The group concluded that it would be a good topic for further collaborations.

#### 4.5 Run-time Architectural Analysis

**Motivation.** Discussions at the working group were concerned with using architectural knowledge in dynamic analysis that is performed during the execution of the system. The

need for run-time analysis is motivated by the two observations:

- On the one hand, the system is not always built according to the model. This is inevitable, since existing generative technologies cannot produce all aspects of the system implementation automatically. Manual implementation inevitably opens the possibility that developers deviate from the model, either by misinterpretation or by overzealous optimization.
- On the other hand, model-based process always realized on the assumptions made during modeling. These assumptions need to be validated at run time. A typical example of such an assumption is failure rates built into the error model of the system [7].

**Design of run-time monitors.** Based on the discussion above, runtime monitoring has two distinct functions:

- Enforce guarantees that have been offered by static architectural analysis done at run time. This includes validation of assumptions that were used to build the model, as well as validation of model parameters.
- Provide error detection and invocation of recovery operations.

Run-time analysis is based on monitoring of the system execution, which means that the system, in addition to the components introduced into the architectural model by the system designer, also implicitly contains components that implement the monitor. The choice of monitoring architecture, to some extent, is determined by the architecture of the system itself.

An important question is how to isolate the observer from the system and minimize, or at least account for, the monitoring overhead. The two common ways of implementing monitors are 1) build the monitor into the control path of the system or 2) run it in parallel with the system. For the first solution, monitoring can be built into the budget for each component during design. Concurrent observers are more powerful, as it is easier for the to maintain a global view of the execution of multiple components. However, their overhead is harder to quantify. Moreover, deployment of concurrent observers presents additional challenges, since an observer has to be synchronized to stable observation points of each component it monitors.

**Open questions and discussions.** In addition to the problem of monitoring of information related to the architectural model, a reverse question can be asked. Many systems already employ run-time observers; for example, for health management [6]. How can we use the available architectural information to improve efficiency of monitoring and reduce overhead. For example, system architecture can be utilized in deciding monitor placement.

The question of quantifying additional robustness is achieved in the system design through run-time monitoring remains an important question to be answered.

#### 4.6 Notion of Time: Physical vs. Real-Time vs Discrete vs Logical time

**Motivation.** The notion of time is central not only in physics, but also in computer-centric systems, where several dimensions are to be combined:

- Semantics: discrete/dense time
- Uniformity: uniform/non-uniform time
- Linearity: linear/non-linear

- Representation: Timed Automata, Petri Nets, Tagged Systems, clocks
- Solving techniques
- Requirements and time

Actually, these dimensions reflect several use cases of time concepts for 1) modeling a system and its semantics then 2) to analyze it.

Hence, *discrete time* can be used to model *logical* time relationships between events (e.g. Lamport clocks, synchronous systems), *discrete events* systems or time scales for simulation. *Dense time* is relevant for physical system (ideal time for Newtonian systems) and is well represented by Timed automata.

Representing a system using a particular class of time (or clocks) is the first aspect. The challenging part is to define consistent solving techniques to assess time-based properties. A timed system is possibly infinite, one needs to map them to an equivalent problem that is finite. Several techniques have been defined: K-induction with monotonic real input, symbolic approaches (region graphs, etc.), explicit model checking (dealing with instants); representation of time using rational or floating-point abstractions.

**Open questions and discussions.** At first, the group questioned the necessity for multiple representations of time. Considering the family of embedded systems, we note that heterogeneous time representations are required to capture the time as seen by the hardware elements of the system, connected to the physical environment; but also logical time for pure software system as a logical abstraction of its behavior. Hence, we need to combine safely these representations, and “meet-in-the-middle” either in a top-down or bottom-up way.

Then, another question is how to map requirements onto this time system. Requirements to be fulfilled by the system are usually expressed in natural language, such as “The pacemaker shall pace the heart at least once per second”. This requires variants of temporal logic (e.g. Timed-CTL) that are usually hard to master.

From these considerations, the group defined the following set of open questions:

- Which analyses are compatible ?
- How to combine different analysis techniques? In a practical way, but not in a purely engineering way !
- What is the role of architecture ? What can be reused/combined?

## 4.7 Patterns for (de)composing and analysing systems

**Motivation.** Patterns for composing and decomposing systems is an essential element for the engineering of complex system. The group reviewed typical strategies to address complexity in systems:

- “Divide and conquer”, where a problem is separated into subproblems being resolved separately. The global problem being solved when all subproblems have a solution;
- “Separation of concerns”, where concerns (e.g. safety and security) are addressed separately, with adapted techniques.

**Open questions and discussions.** One interesting question answered by the group is the relationship between well-known engineering practice and architectures. Architecture is about defining relationships between elements.

We note there are actually many elements to consolidate through an architecture.

“Architecture of structure” focuses on the system decomposition into subsystems, their interfaces, connections, etc. It is also concerned by extension points for later system evolution.

“Architecture of relations” focuses on the comprehension of a system intrinsic nature: functional, safety, reliability, physical, cost, etc.

Both architectures follow a similar “divide and conquer” pattern, the main distinction being that interfaces are more natural to separate than concern. Security can be seen as a particular view of the global system, while divide and conquer applies recursively to (sub)systems elements in an orderly way.

Hence, the challenge is to know how a particular concern can be mapped onto the architecture. Keeping the example of security, it is a global concept that emerges from the combination of all system blocks. The architecture details how basic elements cooperate to provide security.

The group made a convincing point that in initial stages of specification, the combination of concerns/components is usually explicit, but that it tends to get lost in later stages. In some sense, the engineering process evolves from a “correct by construction” paradigm to a “construct by correction” one.

We concluded that system architecture but also architecture modeling framework should guide the designer back on track to follow only a “correct by construction” path.

## 5 Summary and Open Challenges

The working group and the concluding session were the occasion of a lively discussion about open problems and roadmap for our community. The diversity of speakers, themes discussed and existing contributions showed that architecture is a central artifact in many processes, tools and methodology.

We summarize some of the discussions we had, and open challenges outlined by the group:

- **Building an architecture** design activities have a goal that may differ in nature: either test an hypothesis, build a full system ready to be deployed. In between, V&V activities must be conducted. Design patterns helped structuring software activities, but fails short to guide architectural designs.  
**Hot topic** the building of architecture requires higher-level patterns to know how to combine elements. At first, combining interfaces/components, but also concerns (e.g. safety, security, ...). The former is well mastered by the industry, the latter is more problematic. Yet, we note that it is the failure to combine concerns that delay most projects, in particular in the embedded domain.
- **Semantics of an architecture** an architecture interconnects element, and give meaning to this assembly. Defining its semantics is usually done in an informal way (natural language), or through the combination of small elements of semantics (timed automata, syntactic elements, etc).  
**Hot topic** we see the emergence of numerous “Model of Computation” (Ravenscar, synchronous, etc), but also semantics for key aspects like time, fault propagation, ... They precisely describe one semantics, but separately. One needs a global view on how to combine them to give a precise meaning to the whole architecture.
- **Architecture analysis as a MDE topic** Architecture Description Language are bound to model-based technology. Hence, many projects around ADL focus on model transformation techniques, mapping one architecture onto numerous analysis tools.  
**Hot topic** the preservation of semantics between the model of a system, and its analytical model (in a scheduling analysis framework, model checker, etc.) is mandatory to



demonstrate that the results is meaningful. Current technologies fail to demonstrate this mapping in a general case.

- **Analysis of systems** several talks and discussions focused on particular analysis techniques, either to address particular properties (time, fault), or technical limitations (combinatorial explosions). We also noted that analysis is usually performed by people who are experts in a particular problem space (avionics, medical) but not in a given design space (timed automata, Petri nets, ...).

**Hot topic** thanks to MDE, one can map architecture to particular analysis, and take advantage of advanced techniques. One needs to complete this mapping with “wizards” to determine when a model is “ready” for analysis, and how to correct it to reach this state. Another hot topic is to determine how to send back meaningful diagnosis to the architectural designer in case an analysis fails.

- **Coupling architecture/analysis** we notes that analysis rely on architecture artifacts, but could also enrich architectures. This has a complex impact that must be evaluated. As it could definitely helps building better architectures faster.

**Hot topic** an analysis could be defined as a contract set on an architecture. There are several “concerns” associated to analysis techniques and tools. An important topic is to build a cartography of these analysis, and their requirement put on architecture. Such map would ease the definition of architecture that is “correct by construction”.

These different topics are already studied by the different participants. A consensus emerged that all these topics must be addressed in uniform way: either vertically, following one concern and one family of analysis; or horizontally, by combining concerns.

Solution to these different problems will address actual shortcomings in many domains: system engineering, software engineering and model-based techniques that are required to address the complexity of embedded systems.

## 6 Bibliography

- 1 A. Al-Nayeem, L. Sha, D. D. Cofer, and S. M. Miller. Pattern-based composition and analysis of virtually synchronized real-time distributed systems. In *Proc. Cyber-Physical Systems (IEEE/ACM ICCPS'12)*, 2012.
- 2 K. Bae, P. C. Ölveczky, A. Al-Nayeem, and J. Meseguer. Synchronous AADL and its formal analysis in Real-Time Maude. In *Proc. ICFEM'11*, volume 6991 of *LNCS*. Springer, 2011.
- 3 K. Bae, P. C. Ölveczky, J. Meseguer, and A. Al-Nayeem. The SynchAADL2Maude tool. In *Proc. FASE'12*, volume 7212 of *LNCS*. Springer, 2012.
- 4 Y. Ben Maïssa, F. Kordon, S. Mouline, and Y. Thierry-Mieg. Modeling and Analyzing Wireless Sensor Networks with VeriSensor. In *Petri Net and Software Engineering (PNSE 2012)*, volume 851, pages 60–76, Hamburg, Germany, June 2012. CEUR.
- 5 J-M. Couvreur and Y. Thierry-Mieg. Hierarchical Decision Diagrams to Exploit Model Structure. In *Formal Techniques for Networked and Distributed Systems - FORTE*, volume 3731 of *LNCS*, pages 443–457. Springer, 2005.
- 6 N. Mahadevan, A. Dubey, and G. Karsai. Architecting health management into software component assemblies: Lessons learned from the arinc-653 component model. In *The 15th IEEE International Symposium on Object/component/service-oriented Real-time distributed computing*, April 2012.



- 7 I. Meedeniya, I. Moser, A. Aleti, and L. Grunske. Architecture-based reliability evaluation under uncertainty. In *Proceedings of the 7<sup>th</sup> International Conference on the Quality of Software Architectures (QoSA 2011)*, pages 85–94, June 2011.
- 8 J. Meseguer and P. C. Ölveczky. Formalization and correctness of the PALS architectural pattern for distributed real-time systems. *Theoretical Computer Science*, 2012. Article in press, <http://dx.doi.org/10.1016/j.tcs.2012.05.040>.
- 9 S. P. Miller, D. D. Cofer, L. Sha, J. Meseguer, and A. Al-Nayeem. Implementing logical synchrony in integrated modular avionics. In *Proc. DASC'09*. IEEE, 2009.
- 10 Y. Thierry-Mieg, B. Bérard, F. Kordon, D. Lime, and O. H. Roux. Compositional Analysis of Discrete Time Petri nets. In *1st workshop on Petri Nets Compositions (CompoNet 2011)*, volume 726, pages 17–31, Newcastle, UK, June 2011. CEUR.
- 11 Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon. Hierarchical set decision diagrams and regular models. In *Tools and Algorithms for the Construction and Analysis of Systems – TACAS*, volume 5505 of *LNCS*, pages 1–15. Springer, 2009.

## Participants

- De-Jiu Chen  
KTH – Stockholm, SE
- Silvano Dal Zilio  
LAAS – Toulouse, FR
- Patricia Derler  
University of California –  
Berkeley, US
- Mamoun Filali-Amine  
Paul Sabatier University –  
Toulouse, FR
- Laurent Fournier  
Rockwell Collins France, FR
- Serban Gheorghe  
Edgewater Computer Systems  
Inc. – Ottawa, CA
- Lars Grunske  
TU Kaiserslautern, DE
- Jérôme Hugues  
ISAE – Toulouse, FR
- Naoki Ishihama  
JAXA – Ibaraki, JP
- Gabor Karsai  
Vanderbilt University, US
- Alexey Khoroshilov  
Russian Academy of Sciences –  
Moscow, RU
- Fabrice Kordon  
UPMC – Paris, FR
- Brian Larson  
Multitude Corp., US
- Bruce Lewis  
US Army AMRDEC, US
- Ivano Malavolta  
Univ. degli Studi di L'Aquila, IT
- Frédéric Mallet  
INRIA Sophia Antipolis, FR
- Pieter J. Mosterman  
The MathWorks Inc. –  
Natick, US
- Viet Yen Nguyen  
RWTH Aachen, DE
- Thomas Noll  
RWTH Aachen, DE
- Peter Csaba Ölveczky  
Univ. of Illinois – Urbana, US
- Paul Petterson  
Mälardalen University –  
Västerås, SE
- Ramon Serna Oliver  
TTTech Computertechnik –  
Wien, AT
- Shin'ichi Shiraishi  
TOYOTA InfoTechnology Center  
USA Inc., US
- Frank Singhoff  
University of Brest, FR
- Oleg Sokolsky  
University of Pennsylvania, US
- Jean-Pierre Talpin  
INRIA – Rennes, FR
- Michael W. Whalen  
University of Minnesota, US

