ISSN 2186-7437

NII Shonan Meeting Report

No. 2016-5

Architecture-Centric Modeling, Analysis, and Verification of Cyber-Physical Systems

Shin Nakajima
Jean-Pierre Talpin
Masumi Toyoshima
Huafeng Yu

March 21 - 24, 2016

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Architecture-Centric Modeling, Analysis, and Verification of Cyber-Physical Systems

Organizers:
Shin Nakajima (NII)
Jean-Pierre Talpin (INRIA)
Masumi Toyoshima (Denso Corporation)
Huafeng Yu (Toyota Info Technology Center)

March 21 - 24, 2016

The term cyber-physical system (CPS) was introduced by Helen Gill at the NSF referring to the integration of computation and physical processes. In CPS, embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. The principal challenges in system design lie in this perpetual interaction of software, hardware and physics.

CPS safety is often critical for society in many applications such as transportation (whether automotive, trains or airplanes), power distribution, medical equipment and tele-medicine. Whether or not life is threatened, failures may have huge economic impact. The development of reliable CPS has become a critical issue for the industry and society. Safety and security requirements must be satisfied by using strong validation tools. Achieving required levels of safety, security and quality of service needs these properties of the system to be formally proved before its deployment. In the past 15 years, development has moved towards Model Driven Engineering (MDE). With MDE methodology, requirements are gathered with use cases, then a model of the system is built that satisfies the requirements. Several modeling formalisms have appeared in the past ten years. The most successful are the executable models - models that can be exercised, tested and validated. This approach can be used for both software and hardware systems.

A common feature of CPS is the predominance of concurrency and parallelism in models. Development of concurrent and parallel systems has traditionally been clearly split in two different families. The first family is based on synchronous models, primarily targeting design of hardware circuits and/or embedded and reactive systems that are often safety-critical. Esterel, Lustre, Signal and SCADE are examples of existing technologies of that nature, and in many places these have been connected with models of environments as required for CPS modeling. The second family addresses more loosely coupled systems, where communication between distributed entities is asynchronous by nature. Large systems are increasingly mixing both types of concurrency: they are structured hierarchically and comprise multiple synchronous devices connected by buses or networks that communicate asynchronously.

In an architectural model, a CPS is represented by a distributed system as entities 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: each component can contain another sub-system with its own set of components and connections between them. An architecture description language for embedded systems, for which timing and resource availability form an important part of requirements, must in addition describe resources of the system platform, such as processors, memories, communication links, etc. Several architectural modeling languages for embedded systems have emerged in recent years, including the SAE AADL, SysML, and UML MARTE.

An architectural specification serves several important purposes. First, it breaks down a system model into manageable components to establish clear interfaces between components. In this way, complexity becomes manageable by hiding details that are not relevant at a given level of abstraction. Clear, formally defined, component interfaces 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 a change in one component to the linked components. More importantly, an architectural model is a repository to share knowledge about the system being designed. This knowledge can be represented as requirements, design artifacts, component implementations, held together by a structural backbone. Such a repository enables automatic generation of analytical models for different aspects of the system, such as timing, reliability, security, performance, energy, etc.

Since all the models are generated from the same source, the consistency of assumptions w.r.t. guarantees, of abstractions w.r.t. refinements, used for different analyses becomes easier, and can be properly ensured in a design methodology based on formal verification and synthesis methods, using notions such as formally defined interfaces and contracts. In most cases, however, quantitative reasoning in architecture modeling and CPS is predominantly parameterized by the dimension of time. In each of the viewpoints, such as software, hardware, or physical phenomena, that an architecture or CPS model refers to, time takes a different form, namely continuous vs discrete, or event-based vs time-triggered. It is therefore of primary importance to mitigate heterogeneous notions of time to support quantitative reasoning in system design, either using a tentatively unified model for it, or by formalizing abstraction/refinement relations from one to another in order to mitigate heterogeneity.

Despite recent research activities in this area, to formally define or semantically interpret architectural models, we observe a significant gap between the state of the art and practical needs to handle evolving 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 semantic interpretation (an abstraction or a refinement) of the source architecture model by the target analytic tool, instead of hard-coding semantics and patterns into the model generator.

The meeting focuses on discussions to answer the following questions.

- How to support modeling in the large, in different dimensions? The architectural description should tie together system requirements, platform requirements, highlight tradeoffs, etc. How to formalize these require-

ments and to assess them with respect to real-time?

- How to provide a better description of the semantics for an architecture description language in a way that is both easier to understand by humans and easier to interpret by a tool? Thus user-friendly semantics should be soundly linked with existing verification tools. We also envision to enhance the model with the certification credits provided by the validation.
- How to handle extension of the semantics to address new demanding architectures like multi-core, new network technologies, and their application to large-scale systems?
- How to model complex architectural frameworks, such as synchronous or time-triggered designs, RAVENSCAR-compliant systems? How to interconnect the architecture modeling of software-intensive embedded systems to physical concerns like energy consumption, electro-magnetic compatibility?
- How to provide sound, cross-domains, timed/quantitative reasoning in system design to integrate inherently heterogeneous component models ?

The meeting also discusses how the notion of CPS has been evolved to look at a wide variety of software problems[1]. Some of them have been studied independently so far. CPS shed light on the fact that these are closely related, which complicates the problems more than before.

- Safety-critical time-sensitive systems
- Cyber security
- Autonomous / adaptive
- Engineering of resilient systems
- Smart integrated system (or IoT)
- Big Data

Furthermore, uncertainty or *Known and Unknown* must be one of the key aspects of CPS. The meeting opens much discussions on issues such as *verification@runtime* and *self-integrating systems*, both of which are new approaches to solving issues in the presence of a certain form of uncertainty.

[1]Thanks to Peter Feiler.

Meeting Schedule

- March 20th, Sunday Evening
  - Welcome Reception
- March 21st, Monday Morning
  - Opening
  - Introduction to the Meeting (Jean-Pierre Talpin)
  - Self Introduction (all participants)
  - Model-based Architecture (John Koo, Naoyasu Ubayashi)
- March 21st, Monday Afternoon
  - AADL, (Julien Delange, Thomas Noll, Alexey Khoroshilov)
  - Integration (Peter Feiler, Bruce Lewis, John Rushby)
- March 22nd, Tuesday Morning
  - Automotive (Masumi Toyoshima, Hafeng Yu)
  - Testing Automotive Software (Toshiaki Aoki, Takashi Kitamura)
- March 22nd, Tuesday Afternoon
  - Model-Checking (Jerome Hugues, Shin Nakajima)
  - Components (Eugenio Villar, Vania Joloboff, Andreas Gerstaluer)
- March 23rd, Wednesday Morning
  - Time (Dionizo de Niz, Gabor Karsai, Frederic Mallet, Jean-Pierre Talpin)
- March 23rd, Wednesday Afternoon
  - Excursion and Dinner in Kamakura
- March 24th, Thursday Morning
  - Wrap-up Discussions (organized by Vania Joloboff)
  - Closing Remarks (Jerome Hugues)

List of Participants

- Toshiaki Aoki, JAIST, Japan
- Julien Delange, SEI/CMU, USA
- Peter Feiler, SEI/CMU, USA
- Andreas Gerstlauer, Univ. Texas Austin, USA
- Jorgen Hansson, University of Skovde, Sweden
- Jerome Hugues, ISAE, France
- Vania Joloboff, LIAMA, China
- Gabor Karsai, Vanderbilt University, USA
- Alexey Khoroshilov, Russia
- Takashi Kitamura, AIST, Japan
- John Koo, ASTRI, Hong Kong
- Bruce Lewis, AMRDEC Research Lab., USA
- Frederic Mallet, Univ. Nice Sophia Antipolis, France
- Shin Nakajima, NII, Japan
- Dionizo de Niz, SEI/CMU, USA
- Thomas Noll, RWTH Aachen University, Germany
- John Rushby, Stanford Research Institute, USA
- Jean-Pierre Talpin, INRIA, France
- Masumi Toyoshima, Denso, Japan
- Naoyasu Ubayashi, Kyusyu University, Japan
- Eugenio Villar, University of Cantabria, Spain
- Huafeng Yu, Toyota ITC, USA

Collections of Abstracts

1. Model-Based Security Analysis : Julien Delange
2. The CPS Meeting of the Domains: Models, Analyses, and Assumptions : Dionisio de Niz
3. Modular Programming and Reasoning for Dealing with Uncertainty in CPS : Naoyasu Ubayashi
4. Architecture-based Comprehensive Timing Analysis for Distributed Cyber-Physical Systems : Gabor Karsai
5. Correctness, Safety and Fault Tolerance in Aerospace Systems: The ESA COMPASS Project : Thomas Noll
6. Modeling, Analysis, and Verification of Cyber-Physical Systems in the Electronic Century : Eugenio Villar
7. From Virtual System Integration to Incremental Life-cycle Assurance : Peter H. Feiler
8. Model-based Analysis of Energy Consumption Behavior : Shin Nakajima
9. Practical Application of Formal Methods to Automotive Systems : Toshiaki Aoki
10. Summary : Vania Joloboff
11. Specification and Prototyping of Cyber-Physical Networks-of-Systems : Andreas Gerstlauer
12. Architectural Models For Self-Integrating Systems : John Rushby
13. Overview of the Architecture Centric Virtual Integration Process : Bruce Lewis
14. Time, Events and Architectures : Jean-Pierre Talpin
15. MARTE/CCSL for Modeling (and Analysis ?) of Cyber-Physical Systems : Frederic MALLET
16. MASIW an AADL-based Toolset for Modeling, Analysis and Verification of Avionics : Alexey Khoroshilov
17. A Need for High-Level CPS Modeling and V&V : Masumi Toyoshima
18. Model-checking for Ravenscar systems : Jerome HUGUES
19. An architecture-centric design analysis and optimization framework for automotive systems : Huafeng Yu
20. Model-Based Systems Engineering for System-Level Design of Cyber-Physical Systems : T. John Koo
21. Testing specifications : Takashi Kitamura

Cyber-Physical (CPS) systems are now software-reliant and evolving at an incredible pace. With the emerging Internet of Things (IoT) ecosystem, these systems are now interconnected to several networks and exposed to potential attackers. This increases the attack surface and, the likelihood of a successful attack that would penetrate the system. Until recently, many security efforts were focused on code analysis, but studies have shown that security is also a matter of good software architecture design and practices. Software architects need to isolate components according to their security domains and criticity using appropriate methods and techniques. In that context, we are arguing that software designers needs a framework that will help them to (1) capture the architecture with security concerns, (2) evaluate security requirements and policy enforcement and ultimately (3) automatically generate the security policy from validated artifacts.

During our presentation we present our research efforts towards a model-based framework for the design and analysis of secure systems. Our methods and tools leverage the AADL architecture modeling language and extend it with security information in order to perform the different activities of the development process, including security policy validation, implementation, and testing. Using the same model throughout development improves the consistency of the development process by avoiding any translation between different—and potentially inconsistent—representations. In addition, automating the generation of implementation and tests avoids the traditional mistakes of manual code production, such as bugs and developers' assumptions about ambiguous requirements. Our presentation will also demonstrate the status of our current efforts, introduce our tools and present the agenda of coming research efforts.

Acknowledgments

DM-0003221

The CPS Meeting of the Domains: Models, Analyses, and Assumptions

Dionisio de Niz
dionisio@sei.cmu.edu

Software Engineering Institute | Carnegie Mellon University

The development of Cyber-Physical Systems (CPS) increasingly demands more and more analysis tools to understand the complex interactions of software and physics. In the automotive industry, for instance, Simulink and Matlab tools are common to analyze the behavior of the engine control systems. The avionics industry takes advantage of other tools to analyze the systems from the aerodynamics, mechanical stress, thermal, timing, and logical perspective to name a few. We called these CPS analyses.

CPS analyses emerge from different scientific domains that have taken advantage of abstractions to eliminate details not relevant to the domain in order to make the analysis tractable. For instance, logical analysis, like model checking, eliminates execution time and considers actions instantaneous. Similarly, analyses make assumptions that, while known to domain experts, typically go poorly documented and rely on humans to verify their validity. For instance, the original rate-monotonic scheduling analysis assumes that a high-priority task will never suspend after it is activated. If it does a lower-priority task may experience more frequent preemptions from the higher-priority task than the analysis considers. Finally, assumptions from analyses from different domains may contradict each other. Unfortunately, due to the independent evolution of these domains, experts in each domain may be unaware of each other assumptions. For instance, a scheduling expert may want to adjust the frequency of the processors to make the system schedulable, but this can change the voltage demand from the batteries, which in turn can invalidate the thermal runaway analysis of the battery management system.

In this talk I will discuss one of the initial solutions to this problems in a scheme we name *Analysis Contracts*. I will also discuss our current status and the challenges we have ahead of us.

Modular Programming and Reasoning for Dealing with Uncertainty in CPS

Naoyasu Ubayashi
Kyushu University, Japan

Embracing uncertainty in cyber-physical systems (CPS) is one of the crucial research topics [2]. Garlan, D. claims that *software engineering is founded on a myth that the computational environment is predictable and in principle fully specifiable* [1]. He argues that we must embrace uncertainty within the engineering discipline of software engineering. There are three types of software development: *Known Knowns*, *Known Unknowns*, and *Unknown Unknowns*. The *Known Knowns*-type corresponds to the development in which uncertainty does not exist. Many studies on this type have been conducted in traditional software engineering research. In the *Known Unknowns*-type, there are uncertain issues in the process of software development. However, these issues are known and shared among the stakeholders including developers and customers. In the *Unknown Unknowns*-type, it is uncertain what is uncertain.

Modularity is one of the important principles in software engineering. Unfortunately, the state-of-the-art module mechanisms do not regard an uncertain concern as a first-class pluggable software module. *Modularity for Uncertainty* is one of the challenges to be tackled by the software engineering research community. If uncertainty can be dealt with modularly in terms of software architecture, we can add or delete uncertain concerns to/from code whenever these concerns arise or are fixed to certain concerns.

We propose a new programming and reasoning style supporting *Modularity for Uncertainty*. Without modular reasoning about uncertainty, a developer has to rely on global reasoning to check whether some properties are satisfied even if uncertain concerns are contained in architectural models or code.

References

[1] Garlan, D.: Software Engineering in an Uncertain World, In *Proceedings of FSE/SDP Workshop on Future of Software Engineering Research (FoSER 2010)*, pp.125-128, 2010.

[2] Perez-Palacin, D. and Mirandola, R.: Uncertainties in the Modeling of Self-adaptive Systems: a Axonomy and an Example of Availability Evaluation, In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. pp.3-14, 2014.

Architecture-based Comprehensive Timing Analysis for Distributed Cyber-Physical Systems

Gabor Karsai
Institute for Software-Integrated Systems
Vanderbilt University
Nashville, TN 37212, USA

Distributed cyber-physical systems (dCPS) are often operating in complex domains where a high degree of dependability is required. Examples include the power grid, networks of satellites, the air traffic control system, and financial trading systems. A crucial characteristics of these systems is that they must satisfy strict timing requirements under all foreseeable operational scenarios and they have to be resilient to faults in the physical system, in the computing and communication hardware and software, and even to cyber-attacks. The problem is made worse by the need to continually improve and update these systems, as more and more functionality is being implemented in software. The design time analysis and verification of such systems is essential.

Experience shows that the theoretical foundations and the needs of pragmatic engineering of dCPS converge to a component-based architecture for such systems where large-scale distributed real-time embedded software applications are built from 'components' that follow a precisely-defined model of computation. The model of computation here defines what a component is, how its operations are executed, and how it interacts with other components. This model is supported by an underlying component framework that implements the run-time support for scheduling and executing component operations. However, this is insufficient for a distributed system where component interactions involve the network with its own physical characteristics and networking protocols. In summary, for the timing analysis of dCPS a comprehensive approach is needed that takes into consideration the software components of the system and all of their interactions, some of which may involve a communication network.

The talk will present an approach that has been implemented and experimentally validated. The approach uses Colored Petri Net for modeling the timing and behavior of a distributed component-based application and uses an extension of Network Calculus to model and analyze the behavior of the communication network. The two analysis methods are integrated and the analysis models are automatically generated from a concrete architecture model of software applications. The analysis is able to answer questions related to the worst-case stimulus to response delays, potential deadlocks and other timing violations in the system. The architecture model is also used to generate infrastructure code for the application, which is then combined with the business logic code for the components provided by developers and executed on a distributed platform. The instrumented platform provided convincing evidence that the analytical results are valid for a specific implementation.

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

Thomas Noll

RWTH Aachen University, Germany
noll@cs.rwth-aachen.de

Abstract

Building modern aerospace systems is highly demanding. They should be extremely reliable, 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 correctness, safety, dependability and performability. Additionally, they often 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 (http://compass.informatik.rwth-aachen.de/) that was initiated by the European Space Agency to overcome this problem, and that provides an integrated approach to system-software co-engineering of on-board computer-based aerospace systems. It features both a tailored modelling language and a toolset for supporting (semi-)automated validation activities. The language is a variant of the standardised Architecture Analysis and Design Language, AADL, and its Error Model Annex. Its major distinguishing aspects are the possibility to describe hardware and software components and their nominal operation, hybrid (and timing) aspects, as well as probabilistic faults and their propagation and recovery. Moreover, it supports dynamic (i.e., on-the-fly) reconfiguration of components and inter-component connections.

Based on a formal semantics that gives a precise interpretation of AADL models, our framework supports a coherent set of specification and analysis techniques such as system simulation, model checking, safety and dependability analysis, and performance evaluation. We show how these techniques can be applied to assess system-level correctness, safety, dependability, and performability properties of on-board computer-based aerospace systems. We also report on industrial case studies that have been carried out in the context of aerospace systems.

Modeling, Analysis, and Verification of Cyber-Physical Systems in the Electronics Century

Eugenio Villar
University of Cantabria

Moore's Law has dominated the (re-)evolution of electronics during the last quarter of the XX century. All the electronic products we use today depend directly or indirectly on the increasing integration capability allowed by semiconductor technology. This evolution has enabled to produce new electronic products with unexpected capabilities just several years before they appear. The smart-phone and the tablet are examples of such products. This evolution still continues with smart-watches, 3D glasses, drones, etc. Due to the pervasive character of electronics, electronic components are becoming fundamental in the improvement of many non-electronic products such as cars, airplanes, medical equipment, domestic appliances, etc. So, for example, in 2030, 50% of the value of a car is expected to be due to the electronics it contains. The influence of electronics goes further affecting most of the service sector. ICT services, e-banking, e-commerce, security, etc. have evolved dramatically as a consequence of the electronic push. A second consequence of Moore's Law affects the business model. All the electronic products, the electronic components in non-electronic manufactured products and services become obsolete in a short time as a new technology node is available able to produce devices with higher performance at the same cost. Paradigmatic examples are the Intel's Tick-Tock and the iPhone evolution each year. The huge investments required to follow Moore's Law increases dramatically the cost of silicon and limits the accessibility to semiconductor fabrication to big players, both Integrated Device Manufacturers (IDM) and large Fabless semiconductor companies.

This business model will change in the short time as Moore's Law reaches an end. If Moore's Law changed the world, its end may have a similar effect. Cyber-Physical Systems of Systems (CPSoS) will dominate the electronics century becoming pervasive in all the aspects of our daily lives. For the first time, the underlying technology will be stable with only incremental improvements in time. This may make it accessible to many new players looking for a competitive advantage in silicon. Investment will move from the initial stages of the value chain to those closer to the final user.

In this new scenario, modeling, analysis and verification of CPSs will have to evolve. The focus should be put on the device, not isolated but as a component in a complex, heterogeneous, distributed network of many other computing devices. Services will be offered by the interaction of functional components deployed in many distributed computing resources of many kind, from small motes, embedded systems and smart-phones to large data centers and even High-Performance Computing (HPC) facilities. Electronic design in this new context should address effectively new requirements. Among them, scalability, reusability, human interaction, easy modeling, fast design-space exploration and optimization, powerful functional and extra-functional verification, efficient handling of mixed-criticality and security, etc. An essential aspect will be the availability of powerful, platform independent SW and HW synthesis tools able to produce automatically efficient implementations of the system model on many different computing resources. In this presentation, the effect of this dramatic change in system design will be discussed. A single-source approach supported by powerful design tools will be proposed. Current results from the European FP7 ConTrex project will be described.

From Virtual System Integration to Incremental Life-cycle Assurance

Peter H. Feiler – phf@sei.cmu.edu

Feb 25, 2016

Challenging problems associated with system software complexity growth are threatening industry's ability to build next generation safety critical embedded systems. Current best practice of building and then testing software-reliant mission and safety critical systems has led to 80% of requirement and architecture design flaws being discovered post-unit test resulting in rework cost exceeding 50% of the total system cost. Contributors to these problems include the growth of software, system integration, and interaction complexity exacerbated by ambiguous, missing, incomplete, and inconsistent requirements.

In this presentation we discuss a strategy for addressing this problem from two perspectives: 1) The use of an architecture-centric virtual integration practice (ACVIP) to discover system-level issues early in the development life cycle; 2) An incremental assurance approach that focuses on improving the quality of requirements and automating incremental verification throughout the life cycle.

ACVIP is based on the SAE International Architecture Analysis and Design Language (AADL) standard. It is a quantitative, architecture-centric, model-based approach enabling virtual integration analysis in the early phases and throughout the lifecycle to detect and remove defects that currently are not found until software and systems integration and acceptance testing. ACVIP is currently being piloted and matured in a multi-year international Aerospace industry initiative called System Architecture Virtual Integration (SAVI) as well as other industry and government projects.

To improve the quality of requirements we focus on coverage of system specifications, quality attributes, and hazards, as well as management of uncertainty in the requirements. To improve the quality of evidence we use compositional verification, and multi-valued logic to automate the planning, execution of verification plans, and management, reporting of assurance evidence. We do so incrementally along three dimensions: one architecture layer at a time, focus on critical requirements/quality attributes and incrementally expand to the full set, and incrementally manage the impact of changes on requirements, architecture design, and verification evidence. We will illustrate the application of this incremental life-cycle assurance approach on real world examples supported by a prototype tool workbench.

DM-0003370

Model-based Analysis of Energy Consumption Behavior

Shin Nakajima, National Institute of Informatics

Cyber-physical systems (CPS) have brought forward scientific challenges to construct dependable software-intensive systems. The systems constitute social infrastructures to support our daily lives, and have strong connections with their outside environment as *embedded systems* do. These systems are different in their shape and capability. Some of them have a common non-functional property regarding their energy consumption behavior, because they are dependent on batteries. The capacity of a battery is limited and, unless charged, decreases monotonically to eventual depletion. Furthermore, the systems are software-intensive so that they implement smart services, in which application programs are indirectly responsible for consuming the battery power. Running a buggy program may result in a large amount of energy unexpectedly. This brings a new technical challenge of eliminating energy bugs (e-bugs) during the development of software-intensive systems.

Model-based analysis methods are finding potential e-bugs in application software designs. Such methods use *a formal model* to account for the behavior of both application programs and functional hardware components. Although hardware components are the direct consumers of battery power, programs are responsible for energy consumption indirectly, because they control the hardware usage. An analysis model must represent discrete behavior of programs and use real-valued variables to account for execution time or consumed energy. Thus, model-based analysis methods are studied in regard to *the co-existence of Booleans and reals*. It is exactly one of the primary characteristics of CPS.

Establishing model-based methods of analyzing energy consumption behavior is mandatory for developing trustworthy cyber-physical systems. We study such a method using a variant of linear hybrid automata as a rigorous model, and then reduce the problem of detecting anomalies in the energy consumption behavior to logic model checking.

Practical Application of Formal Methods to Automotive Systems

Toshiaki Aoki
Japan Advanced Institute of Science and Technology

The safety and reliability of automotive systems are becoming a big concern in our daily life. Recently, a functional safety standard which specializes in automotive systems has been proposed by the ISO. In addition, electrical throttle systems have been inspected by NHTSA and NASA due to the unintended acceleration problems of Toyota's cars. In light of such recent circumstances, we are researching practical applications of formal methods to ensure the high quality of automotive systems. We have several joint projects with our industrial partners. In this presentation, we firstly introduce the overview of those projects. Then, we focus on one of them, the verification of an automotive operating system with model checking and testing.An operating system which we focus on is the one conforming to the OSEK/VDX standard.

JAIST and DENSO started a joint research project in 2006. DENSO develops automotive software using OSs which are provided by the other companies. We examined the feasibility of applications of formal methods at this point. Then, we decided to apply formal methods to a commercial OS whose target CPU is V850. Renesas Electronics Corporation(REL) which develops this OS and CPU joined this project in 2009. We call the OS 'REL OS' below. REL OS has been already released and used in a current series of cars at this time. It is needless to say that traditional methods have been applied to REL OS in order to check it then. Our aim is to achieve higher quality of the OS for next series of cars by applying formal methods.

This presentation shows a case study that model checking is applied to a commercial OS, that is, REL OS. REL OS is too complicated to convince us that it correctly performs for any application. We adopted exhaustive verification techniques to check REL OS. We have conducted exhaustive testing based on a design model which was exhaustively verified by model checking. As a result, we acquired the confidence that REL OS correctly performs for any application although no new bug was found. The model checking and testing were more exhaustive and reliable than the traditional methods. Such combined model checking and testing are appropriate to convince us of the correctness thanks to their exhaustive nature.

Summary Vania JOLOBOFF

For the past 10 years I have been working on virtual prototyping, whereas I previously worked in compiler, operating systems and distributed systems.

I have worked on virtual prototyping of complete embedded systems from the prospective of the architect and software development engineers. The main issues from this prospective are (i) the simulation speed that must be fast enough to cope with Hardware In the Loop or Human In the Loop and (ii) the different methods that can be used to ensure the system is working properly, according to the requirements, whether they are automated tests or formal proofs using various techniques. In order to achieve that, we have set up a virtual prototyping framework, SimSoC, which is distributed as open source on INRIA forge.

The goal of SimSoC is “Full System Simulation”, that is, the system virtual prototype boots exactly like the real device with exactly the same embedded software, including the operating system. SimSoC applications can boot Linux operating system on fully simulated hardware such as the multi-core Freescale 8641D SoC.

The cornerstone of SimSoC is a library of hardware simulation modules, that include Instruction Set Simulators (ISS) for ARM and MIPS and Power architectures. All of the SimSoC ISS rely on the same technology, namely a Dynamic Binary Translation that works in several steps. First the binary code is translated in a data structure representing the basic blocks. Second the basic blocks are individually optimized for faster simulation. An observer mechanism measures the frequency usage of basic block to detect the 'hot spots' and those hot spots are translated into native code, via the LLVM JIT compiler. A garbage collector is included to detect code that has become stale. SimSoC simulation speed can reach over 500 Millions of Instructions per second. SimSoC also contains tools to debug the virtual prototype. A native debugger such as GDB can be connected to the virtual prototype to debug the embedded software. Also a specific version (for the Power Architecture e200 only) has been developed with Approximately Timed measurements so that software developers can obtain performance evaluation of the embedded software on the real hardware with a good approximation. It also contains a module to interface with the real network so that virtual devices can be simulated on a computer, using their own IP address and TCP/IP software.

We have been working more recently on techniques to proof correctness of the application. A large effort has been undertaken to prove that the ISS coded in C for the ARM architecture is really simulating the ARM processor from a formal definition, using the Coq theorem prover.

Since about one year, I have been collaborating with Pr Frederic Mallet from University of Nice to introduce a runtime verification component in SimSoC based on logical clocks. There exist property specification languages such as PSL but they require that the properties are known at compile time of the virtual prototype. But in real life practice, engineers discover properties that entail from the requirements or that should be enforced by software due to hardware constraints while debugging the system. We are working towards a system where the engineers can define new properties and check them dynamically without recompiling the virtual prototype and without storing huge trace files.

Specification and Prototyping of Cyber-Physical Networks-of-Systems

Andreas Gerstlauer
The University of Texas at Austin

A key aspect of future embedded and cyber-physical systems (CPS) is networking of traditionally disconnected computer systems. This poses new fundamental design challenges. Machine-to-machine communication comes with inherent uncertainties, such as data losses, yet systems have to perform under tight performance guarantees. Furthermore, network-level mapping, scheduling and offloading of computations comes with non-obvious tradeoffs that will greatly influence overall application performance and power consumption. Systematically exploring such design spaces requires new approaches for network/system co-design across communication and computation boundaries.

In this talk, we present our early work on developing the modeling foundations for a novel network-level co-design science of future networks-of-systems (NoS). This includes: (1) formal models of computation and communication (MoCC) for NoS specification, including associated analysis and synthesis, and (2) fast yet accurate NoS co-simulation models for early virtual prototyping and rapid design space exploration. The former are based on extensions of existing deterministic datatflow models to expose inherent network uncertainties and explicitly capture associated application adaptivity and reactivity. The latter are based on adapting existing technologies for host-compiled simulation of multi-processor/-core system-on-chip (MPC-SoC) platforms. In such models, source-level application code is natively compiled onto the simulation host and first back-annotated with performance estimates obtained either from accurate reference simulations or by employing novel machine-learning based approaches for source-level power and performance prediction. Back-annotated code is then wrapped into light-weight operating system and processors models that further integrate into standard, SystemC-based transaction-level modeling (TLM) and network simulation backplanes. Such models have been shown to simulate at close to native execution speeds (in excess of 1000MIPS) with near cycle accuracy (less than 5% timing and energy error). Furthermore, by integrating with floorplanning and thermal models, an exploration of real application code running on complex NoS across a full range of performance, energy, reliability, performance and thermal (PERT) metrics becomes possible.

Bio: Andreas Gerstlauer is an Associate Professor in Electrical and Computer Engineering at The University of Texas at Austin. He received his Ph.D. in Information and Computer Science from the University of California, Irvine (UCI) in 2004. Prior to joining UT Austin in 2008, he was an Assistant Researcher in the Center for Embedded Computer Systems (CECS) at UC Irvine, leading a research group to develop electronic system-level (ESL) design tools, commercial derivatives of which were used at the Japanese Aerospace Exploration Agency (JAXA), NEC Toshiba Space Systems and others. Dr. Gerstlauer is co-author on 3 books and more than 80 publications. He has presented in numerous conference and industrial tutorials, is an editor for ACM TECS and TODAES journals, and has served as the topic, track or program chair of major international conferences such as DAC, DATE, ICCAD and CODES+ISSS. His research interests include system-level design automation, system modeling, design languages and methodologies, and embedded hardware and software synthesis.

Architectural Models For Self-Integrating Systems

John Rushby

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025 USA

Systems necessarily interact with their neighbors through the effect each has on the environment of the others (so-called *stigmergic* interactions), particularly the "plant"—for example, several medical devices may be attached to the same patient. Unmanaged interactions can be deleterious (e.g., gridlock in California, where cars at a green light cannot proceed because traffic is backed up from the uncoordinated red light further ahead) and it is better if systems are *open* to interactions, so these can be coordinated. Furthermore, systems should be *adaptive* so they can adjust their behavior to their circumstances. We then have *systems of systems*, which can exhibit desired and positive behavior, but it is also possible that normal or adjusted (or faulty) behavior by one component system adversely affects others, leading to *emergent misbehavior*.

Not all interactions can be planned ahead of time so systems need to *self integrate* as they encounter other systems at runtime. But we want the resulting integrations to guarantee properties such as safety, so self-integration needs to construct or update an *assurance case*.

One way accomplish all this is for the self-integrating systems to exchange models of their architecture, behavior, and local assurance case: this is (*safety*) *models at runtime* (M@RT and SM@RT). The models are used to guide adaptation: for example, one component system may synthesize a monitor or wrapper for its interaction with another to eliminate some class of behaviors that violate its assumptions. Thus, trustworthy self-integration requires autonomous adaptation, synthesis, and verification at integration time, and this means that embedded automated deduction will be the engine of integration.

I hope to stimulate discussion on the kinds of architectural and other models that can best support trustworthy self integration.

References

1. John Rushby. Trustworthy self-integrating systems. In Nikolaj Bjørner, Sanjiva Prasad, and Laxmi Parida, editors, *12th International Conference on Distributed Computing and Internet Technology, ICDCIT 2016*, Volume 9581 of Springer-Verlag LNCS, pages 19–29, Bhubaneswar, India, January 2016.
2. Mario Tokoro. *Open Systems Dependability—Dependability Engineering for Ever-Changing Systems*. CRC Press, 2013.
3. Mario Trapp and Daniel Schneider. Safety assurance of open adaptive systems—a survey. In Nelly Bencomo, Robert France, Betty H.C. Cheng, and Uwe Assmann, editors, *Models@Run.Time: Foundations, Applications, and Roadmaps*, volume 8378 of Springer-Verlag LNCS, pages 279–318, 2014.

Title – Overview of the Architecture Centric Virtual Integration Process

Bruce Lewis

US Army Research Development and Engineering Laboratory

There is a significant need for Architecture-Centric Modeling, Analysis, and verification within the aviation domain. These techniques are being applied in early experiments sponsored by our research organization and led by the author. Our approach strongly leverages the Architecture Analysis & Design Language (AADL), an engineering process, the Architecture Centric Virtual Integration Process (ACVIP), for the application of the AADL, and a number of analysis tools in the domains of utilization, latency, timing, scheduling, safety, fault tolerance, and security and includes generation of the system configuration files and glue code to integrate the system described in the architecture specification and verified with the models.

My presentation includes the justification for why an analytical process for architecture analysis is required based on the very high costs of system integration in our domain. It demonstrates the significant gap we now have in the development process that is driving these high costs and the need for a multi-domain architectural modeling environment to overcome the issues we are experiencing. I provide a short history of development of the AADL and these methods through DARPA research to the current capability and AADL standard documents and annexes. I discuss the coverage of the AADL for system, software and computer hardware and the modeling methods we are using in the context of integrated analyses against the system architecture specification. Concepts of model refinement and verification along with consistency analysis based on the System Architecture Virtual Integration (SAVI) program are included. Our process is incremental, evaluating integration feasibility and verifying requirements, constraints and assurance cases at each stage throughout development. I will discuss our projects and international projects that are taking a similar analytically driven, architecture-centric approach and our roadmap for future research. I will also provide some information based on return on investment calculations for aviation system development.

Time, Events and Architectures.

Jean-Pierre Talpin, INRIA

A cyber-physical (or reactive, or embedded) system is the integration of heterogeneous components originating from several design viewpoints: reactive software, some of which is embedded in hardware, interfaced with the physical environment through mechanical parts. Time takes different forms when observed from each of these viewpoints: it is discrete and event-based in software, discrete and time-triggered in hardware, continuous in mechanics or physics. Design of CPS often benefits from concepts of multiform and logical time(s) for their natural description. High-level formalisms used to model software, hardware and physics additionally alter this perception of time quite significantly.

In model-based system design, time is usually abstracted to serve the purpose of a single of many design tasks: verification, simulation, profiling, performance analysis, scheduling analysis. For example, in non-real-time commodity software, timing abstraction such as number of instructions and algorithmic complexity is sufficient: software will run the same on different machines, except slower or faster. Instead, in cyber-physical systems, multiple recurring instances of meaningful events may create as many dedicated logical clocks, on which to ground modeling and design practices.

Time abstraction increases efficiency in event-driven simulation or execution (i.e. SystemC simulation models try to abstract time, from cycle-accurate to approximate-time, and to loosely-time), while attempting to retain functionality, but without any actual guarantee of valid accuracy (responsibility is left to the model designer). Functional determinism (a.k.a. conflict-freeness in Petri Nets, monotonicity in Kahn PNs, confluence in Milner's CCS, latency-insensitivity and elasticity in circuit design) allows for reducing to some amount the problem to that of many schedules of a single self-timed behavior, and time in many systems studies is partitioned into models of computation and communication (MoCCs). Multiple, multiform time(s) raises the question of combination, abstraction or refinement between distinct time bases or domains. The question of combining continuous time and discrete logical time demands formal reasoning in simulation. While timed reasoning takes multiple forms, there is no unified foundation for reasoning about multi-form time in system design.

The objective of project TEA is to define formal methods and models for timed quantitative reasoning, or, put simply, for time reasoning, in embedded and cyber-physical system design. Formal time models and calculi constitute a powerful mean to revisit common domain problems in real-time system design, such as functional determinism, time predictability, memory resources predictability, real-time scheduling, mixed-criticality and power management; yet from the perspective gained from inter-domain timed and quantitative abstraction or refinement relations. By this regained focus, we aim at delivering better tooled methodologies for virtual prototyping and virtual integration of embedded architectures.

MARTE/CCSL for Modeling (and Analysis ?) of Cyber-Physical Systems

Speaker: Frédéric MALLET, Université Nice Sophia Antipolis

Abstract:

Cyber Physical Systems (CPS) combine digital computational systems with surrounding physical processes. Computations are meant to control and monitor the physical environment, which in turn affects the computations. The intrinsic heterogeneity of CPS demands the integration of diverse models to cover the different aspects of systems. The UML proposes a great variety of models and is very commonly used in industry even though it does not prescribe a particular way of using those models together. The MARTE profile proposes a set of extensions to UML in a bid to allow for the modeling of real-time and embedded systems (RTES). Yet CPS are a wider class of systems than mere RTES. Hence a legitimate question arises as whether MARTE can be used for CPS as well. This talk discusses some possible uses of MARTE to model CPS and uses logical clocks as a way to bring together the different models.

Logical clocks appear as interesting abstractions to coordinate both digital and physical models. The use of MARTE/CCSL, which gives a concrete syntax to describe logical clock specifications, opens questions on what kind of verification is possible and what needs to be added to tackle cyber physical systems in particular. On this part, in particular, some possible extensions of CCSL with stochastic constructs are discussed in a bid to connect to existing verification frameworks, including stochastic model-checking.

Frédéric Mallet: MARTE/CCSL for Modeling Cyber-Physical Systems. SyDe Summer School 2015: 26-49 (2015)

Matias Ezequiel Vara Larsen, Julien DeAntoni, Benoît Combemale, Frédéric Mallet: A Behavioral Coordination Operator Language (BCOoL). MoDELS 2015: 186-195 (2015)

Amani Khecharem, Carlos Gomez, Julien DeAntoni, Frédéric Mallet, Robert de Simone: Execution of heterogeneous models for thermal analysis with a multi-view approach. FDL 2014: 1-8

Jing Liu, Ziwei Liu, Jifeng He, Frédéric Mallet, Zuohua Ding: Hybrid MARTE statecharts. Frontiers of Computer Science 7(1): 95-108 (2013)

MASIW – an AADL-based Toolset for Modelling, Analysis and Verification of Avionics

Alexey Khoroshilov (Institute for System Programming of the Russian Academy of Sciences)
khoroshilov@ispras.ru

Growth of modern avionics systems makes design of such systems impossible without involvement of automation. MASIW is an AADL-based toolset for design of modern avionics systems developed by ISPRAS in collaboration with GosNIIAS. The toolset provides both a generic platform for design and analysis of architecture models and a specialized solution for the particular domain of avionics systems. It supports creation, editing and management of AADL models in both textual and graphical notation. Also MASIW provides various features for analysis and synthesis of AADL models.

Analysis capabilities include

- a checker of static structural constraints such as resource sufficiency, interface consistency, usage domain rules, etc.
- specialized analyzer of AFDX networks aimed to statically estimate latencies, buffer usage, etc.
- a simulator of AADL models augmented with behaviour specification in AADL Behaviour Annex notation or in Java (also simulation can be combined with execution of actual applications in virtual environment managed by the simulator).

Synthesis capabilities include schedule generation for a particular processor module as well as automatic building of assignment of hardware platform resources to software components in accordance with all requirements formalized in the architecture model.

Finally, MASIW provides a framework for generation of configuration data from architecture models. Currently it is used for generation of configuration tables for ARINC-653 operating systems and for AFDX switches and endpoints.

During development of the toolset we met a number problems with semantics for an architecture description language. Generally, modelling languages are needed for both following usages:
• to express some ideas by some author (usually, a human) and
• to perceive these ideas by some particular interpreter (which can be a human or a machine).

These two usages of modelling languages contradict with each other. The more powerful a modelling language is, the easier to write models for author and the harder they are perceived by interpreters (in particular, the harder to implement a machine interpreter). Vice versa, the less powerful the modelling language, the easier to interpret it and the harder to use it by a model author.

If the intended interpreter of a modelling language is a machine, the language have to have formal semantics. Sometimes modelling languages are used to solve a lot of tasks some of which are not automated yet. In such cases those parts that are intended to be used only by human may be left not so formalized. Generally this situation is normal but since models are tending to become more and more complicated, covering of analysis tasks by automation becomes more and more essential.

To analyse architecture models by a machine, domain-induced formal interpretations have to be defined. Intermediate models can be used to reduce effort of formalization and implementation of such interpretations. All transformations of such interpretations are formal. These formal transformations should be checked for adequacy to informal semantics which is defined by normative documents.

We believe that such formal transformations have to be more actively used within architecture language specifications to make architecture languages formally interpretable which means that is it easily and robustly analyzable by a machine. This allows to build analyzers for user-defined characteristics which are formalized and can be implemented in different instruments with the same precise semantics.

A Need for High-Level CPS Modeling and V&V

Masumi Toyoshima

In my understanding, the most distinguished characteristics of CPS are the tight coupling with physical world and the feedback loop. One example of such CPS is ADAS and AD systems in automotive domain (Advanced Driver Assistance Systems, Automated Driving).

When we think about developing an AD system, we model the CPS feedback, sensing & computing & actuating with much environment information including pedestrians, lanes, tires, road surfaces, etc. While doing this, we always encounter the inherent uncertainty including sensing errors, lanes hard to see, the next move of pedestrians or other vehicles, etc. Actually in the development phase of control modeling and simulation, it is not so difficult to build models which include these uncertainties, for example with probabilities and conduct simulation and validate by human decision. But we need to implement these control logics on embedded computers which act deterministically and they must be verified against safety requirement, which sometimes is not described with probability or quantity.

Several works trying to define the criteria of autonomous machines including automated driving systems are running in academia and some governments. I expect they will be high-level definitions of criteria from us developers point of view because of the tight connection with the society. So I think architectural level integration and verification will be much more important for my purpose.

Model-checking for Ravenscar systems

Jerome HUGUES

Safety-critical software engineering such as hard real-time system is a challenging research topic. Recent development methodologies suggest various solutions based on abstraction and automation for more trusty system construction. Formal verification in early phases of development process, seems an import solution to reduce risk of errors. Yet system should be well specified to obtain useful analysis results. In this talk, we propose a formal mapping for an abstraction of real- time system based on LNT. We focus mainly on scheduling analysis with fixed/unfixed priority algorithms and asynchronous communication between periodic and sporadic tasks. Experimentation illustrates results with three famous case studies. In addition, we define an automatic model verification to validate the applicability of our proposal.

This talk is based on joint work with Hana Mkaouar and Bechir Zalila (ENIS)

An architecture-centric design analysis and optimization framework for automotive systems

Huafeng Yu

TOYOTA InfoTechnology Center, USA

Huafeng.yu@us.toyota-itc.com

(Joint work with UC Riverside)

In automotive domain, architecture design was not considered to be essential, thus it is rarely formalized and analyzed in the design stage in conventional automotive design processes. Consequently, it generally leads to a manual, error-prone, time-consuming architecture exploration and validation. To avoid this problem, formalization, formal reasoning, and early-phase exploration are required, along with explicit quality attributes associated with particular architectural entities. Currently, architectural aspects of the system are not well expressed by general modeling languages, like UML or SysML. Domain-specific architecture description languages, such as AADL, AUTOSAR and EAST-ADL were therefore proposed for embedded systems, especially aviation and automotive systems. With these languages and their associated methodologies, a system-level design, considering architecture and behavior, as well as an architecture-centric analysis and optimization is becoming a promising solution to promote virtual engineering solutions for automotive embedded control systems.

In this talk, an architecture-centric system-level design framework will be presented. The proposed framework and its work flow are also based on platform-based design. During the design process, software and hardware are abstracted as software and hardware models. The meeting-in-the middle process is then started to map the instances of the top platform (i.e., software model) with constraints into the instances of the lower platform (i.e., hardware model) with appropriate constrains. During this process, we analyze and evaluate various options on automotive software and hardware architecture, including selection of computational units (ECUs and possibly GPUs and FPGAs) and communication protocols, generation of software tasks, and mapping and scheduling of software tasks on hardware platforms. We also consider multiple metrics and their trade-offs, including performance, fault tolerance, security, extensibility, etc. We will target applications of advanced driver assistance systems, as well as autonomous driving and vehicular networks, which typically have high data volume and stringent timing requirements.

H. Yu, Software Challenges for Automotive Cyber-Physical Systems, In newsletter of IEEE Technical Committee on Cybernetics for Cyber-Physical Systems (CCPS), 1:7-11, Feb, 2016.

B. Zheng, P. Deng, R. Anguluri, Q. Zhu and F. Pasqualetti, Cross-Layer Codesign for Secure Cyber-Physical Systems, to appear in the IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems (TCAD), 2016.

H. Yu, P. Joshi, J.-P. Talpin, S. Shukla, and S. Shiraishi, The challenge of interoperability: model-based integration for automotive control software, in the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 58:1--58:6, San Francisco, June, 2015.

Model-Based Systems Engineering for System-Level Design of Cyber-Physical Systems

T. John Koo, Ph.D.
Director, Cyber-Physical Systems
Hong Kong Applied Science and Technology Research Institute (ASTRI)

CPS (Cyber-Physical Systems) defined as the future generations of embedded ICT systems deeply connected between the information world and the physical world opens up a wide range of innovative applications and service in the domains of automotive, aerospace, health and energy as well as smart city. In the Fourth Industry Revolution, Industry 4.0, CPS is fundamentally changing the landscape of the manufacturing industry. We focus particularly on Virtual Prototyping, which deploys Model-Based Systems Engineering (MBSE) principle for supporting the system-level design and emulation of the complex system dynamics and the evaluation of the overall system performance prior to constructing any physical prototypes in order to reduce design iterations and optimize for higher levels of performance and reliability.

2016/03/10

Takashi Kitamura (AIST)

Testing specifications.

Formal specifications (i.e., specifications described by languages with formal semantics) play an important role in developing safety-critical systems. As Cyber Physical Systems (CPSs) underpins our daily-life and they will do so even more in the future, the role of formal specifications in developing CPSs will become more and more important. As the importance of formal specifications increases, their correctness has an important meaning. The main approaches to the correctness of formal specifications are model checking and formal proofs. These two techniques are indeed powerful in a sense that they can formally prove the correctness; the former by fully automatically and the latter by an interactive manner. However, both techniques also have disadvantages; that of model checking is scalability, which is often known as the state explosion problem, while that of the formal proof is its high cost induced by the fact that the technique requires efforts of highly-skilled experts on this technique. Indeed, it is believed that these have been main obstacles of wider prevalence of the techniques in industry. As these techniques have their own disadvantages which are too big in real world development, testing may become an yet another alternative approach. Although testing cannot prove the correctness like model checking and formal proof, it can also avoid the scalability and the cost problem; it is more scalable than model checking, and cheaper than formal proofs. Moreover, the literature of testing has found certain effectiveness of the techniques. Accordingly, testing may be used for guaranteeing the correctness at a certain level. In this meeting, I want to discuss about the possibility and effectiveness of guaranteeing the correctness of specifications by testing.