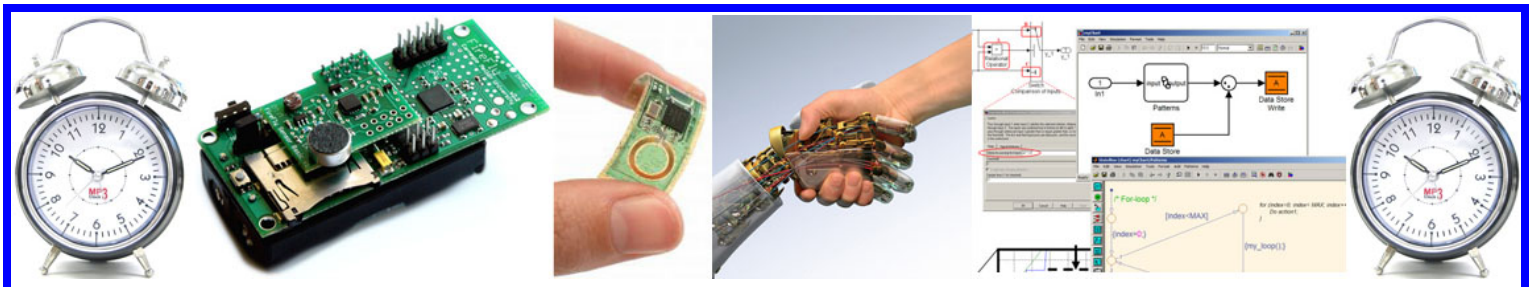


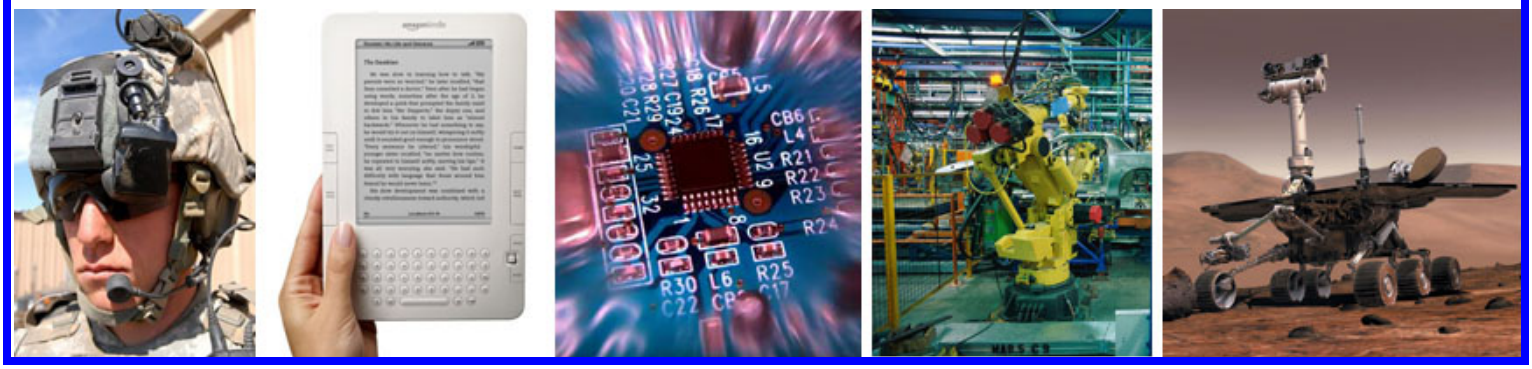
2nd Workshop on

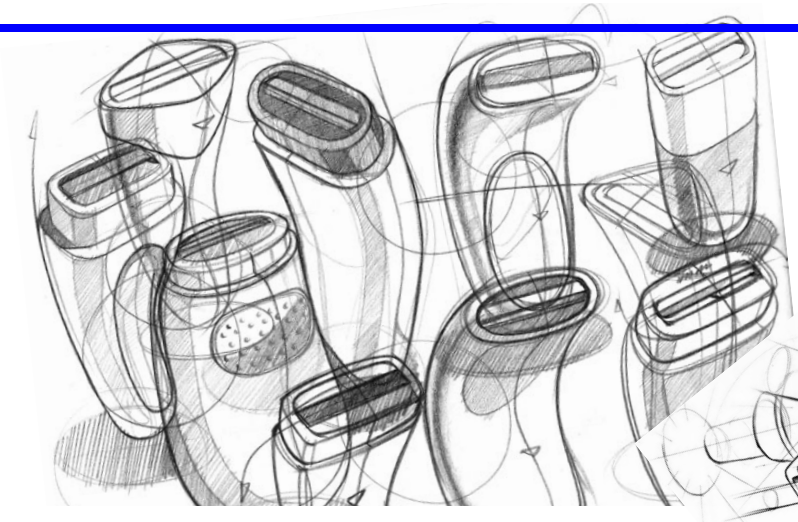
Analytic Virtual Integration for Cyber-Physical Systems

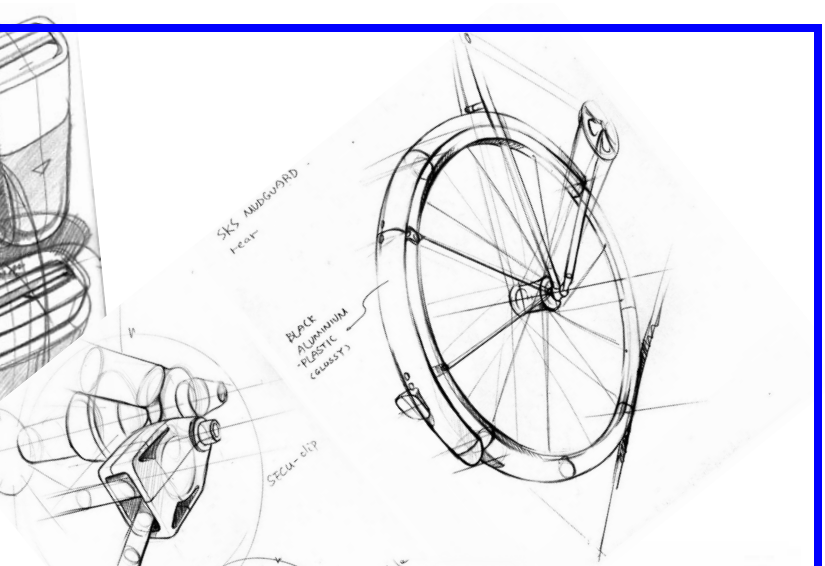
IEEE AVICPS 2011

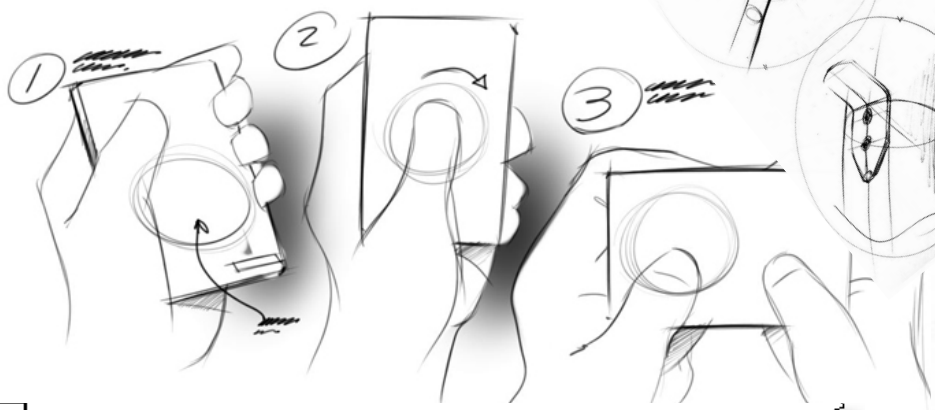
In conjunction with IEEE RTSS, Vienna, Austria

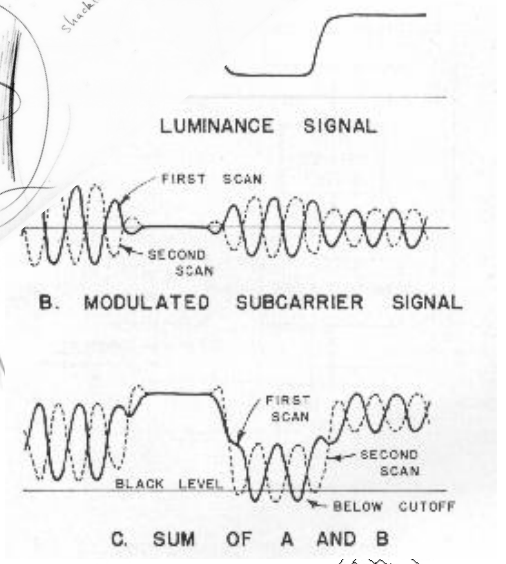
November 29th, 2011

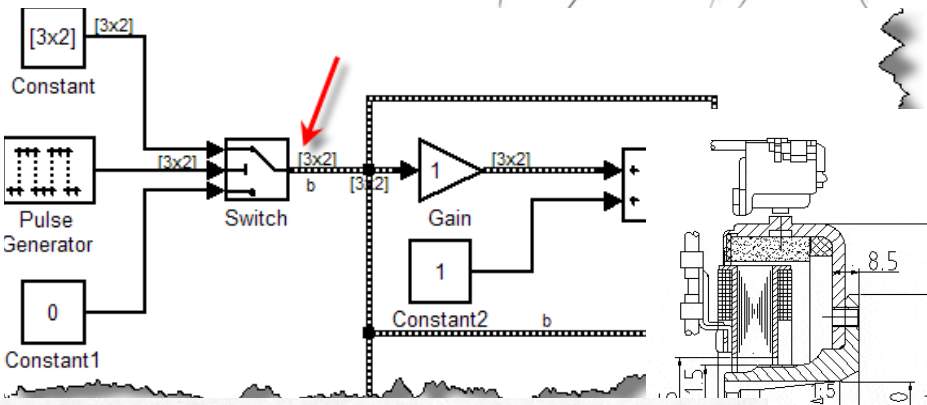


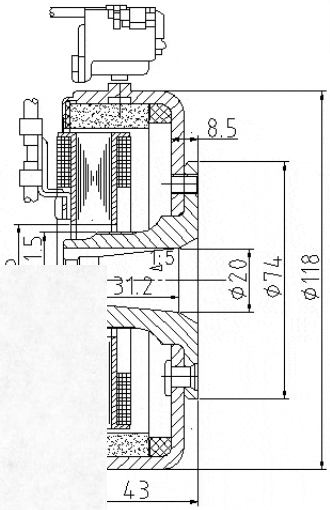


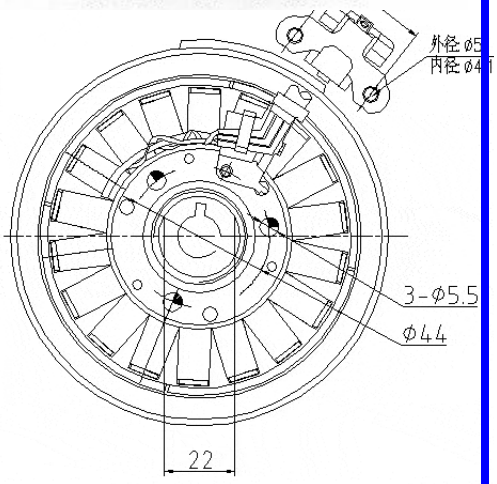


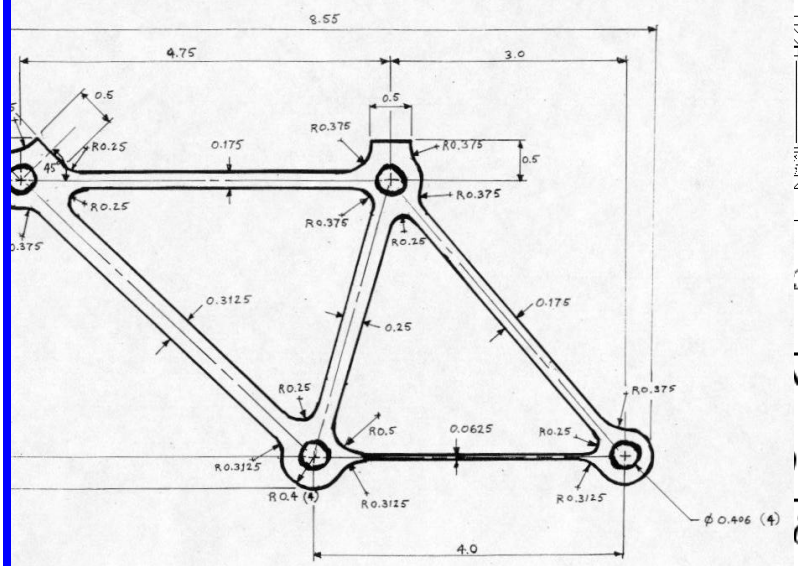












$$x_i \qquad c\,\partial x_k\,\partial t \qquad c^2\ \partial t^2 \qquad c$$

$$\nabla^2 A_k + \frac{1}{c}\frac{\partial}{\partial x_k}\frac{\partial \phi}{\partial t} + \frac{1}{c^2}\frac{\partial^2 A_k}{\partial t^2} = \frac{4\pi}{c} J$$

$$\frac{\partial^2 A_k}{\partial t^2} + \frac{\partial}{\partial x_k}\left(\vec{\nabla}\cdot\vec{A} + \frac{1}{c}\frac{\partial \phi}{\partial t}\right) = \frac{4\pi}{c} J$$

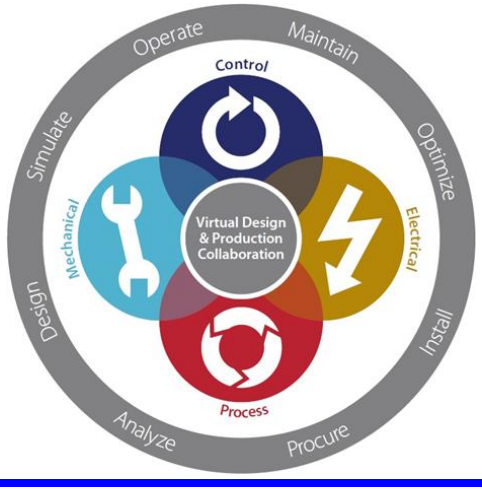
$$\frac{1}{c^2}\frac{\partial^2 \vec{A}}{\partial t^2} + \vec{\nabla}\left(\vec{\nabla}\cdot\vec{A} + \frac{1}{c}\frac{\partial \phi}{\partial t}\right) = \frac{4\pi}{c}$$

Analytic Virtual Integration for Cyber-Physical Systems

The goal of the **Analytic Virtual Integration of Cyber-Physical Systems** (AVICPS) workshop is to explore architecture design patterns, tools and the theoretical analytical foundations for creating common system-wide composition models where key properties can be studied and guarantees provided before the start of actual development. Of particular interest are the case studies on the challenges of expressing the properties of the final product in terms of component properties and the architecture that governs their interactions. Both solutions and/or open problems are welcome.

This workshop focuses on analytical system composition technologies that include:

1. System level schedulability optimization technologies that support the combinatory optimization of task allocation, I/O and network traffic routing
2. A quantitative and early analysis of the system architecture performance in an end-to-end fashion, deriving perhaps even the worst/best/average case behavior for the entire platform and new hardware abstractions. In fact, existing task/system models reason at levels that are well abstracted away from real details of hardware components (e.g. multicores, memory architectures, I/O, network-on-chip, etc.). They also are unable to cope with workloads that are beyond the capabilities of traditional computational resources (e.g. video streams, weather data, GPS, etc.)
3. Fault tolerance technologies against combined cyber faults and physical system disturbances
4. Safety analysis such as model checking for mixed criticality CPS applications, for example, flight management systems and safe medical devices plug and play (MDPnP)
5. Security protocol development and verification techniques for CPS applications
6. Models for describing/quantifying the environments where such systems must operate



Program Committee:

Kirstie Bellman, Aerospace Corporation, USA
Ken Butts, Toyota, USA
Marco Caccamo. University of Illinois at Urbana-Champaign (UIUC)
Julien Delange, European Space Agency (ESA), The Netherlands
Jorgen Hansson, Chalmers University, Sweden
Boudewijn Haverkort, University of Twente and Embedded Systems Institute
Jerome Hugues, Institut supérieur de l'aéronautique et de l'espace (ISEA)
Mirko Jakovljevic, TTTech, Austria
Bruce Krogh, Carnegie Mellon University, USA
Sibin Mohan. University of Illinois at Urbana-Champaign (UIUC), USA
Dionoso de Niz. Software Engineering Institute (SEI), USA
Thomas Noll, RWTH Aachen, Germany
Rodolfo Pellizzoni, University of Waterloo, Canada
Eric Senn, Université de Bretagne Sud (UBS), France
K. C. Shashidhar, Max Planck Institute for Software Systems, Germany
Oleg Sokolosky, University of Pennsylvania (Penn), USA

Program Co-Chairs:

Rahul Mangharam, University of Pennsylvania (Penn). USA
Peter Feiler, Software Engineering Institute (SEI), USA

Workshop Agenda

8:30 - 9:00am Registration and Coffee

9:00 - 9:10am Welcome and Kick-off

9:10 - 10:00am Keynote:
Industry Perspectives on Complex System Integration Issues
Dr. David A. Redman, Director of the Aerospace Vehicle Systems Institute (AVSI)

10.30 – 11:00am Coffee break

10:30 - 12:00pm: Session I
FUSED: A Tool Integration Framework for Collaborative System Engineering
Mark Boddy, Martin Michalowski, August Schwerdfeger, Hazel Shackleton and Steve Vestel
(martin.michalowski@adventiumenterprises.com)

A Design Framework for Model-based Development of Complex Systems
Hristina Moneva, Roelof Hamberg and Teade Punter (hristina.moneva@esi.nl)

Analytic Virtual Integration of Cyber-Physical Systems & AADL: Challenges, Threats and Opportunities
Jerome Hugues (jerome.hugues@isae.fr)

From Abstract Component Descriptions to Timed I/O-Interfaces in AUTOSAR
Stefan Neumann and Sebastian Wätzoldt (stefan.neumann@hpi.uni-potsdam.de)

12:30 - 2.00pm Lunch

2:00 - 3:30pm Session II
Quantitative Fault Propagation Analysis for Networked Cyber-Physical Systems
Linda Briesemeister, Grit Denker, Daniel Elenius, Ian Mason, Srivatsan Varadarajan, Brendan Hall, Devesh Bhatt, Gabor Madl and Wilfried Steiner (linda.briesemeister@sri.com)

Integrating Sleep Scheduling and Compressed Sensing in Sensor Networks
Debojit Dhar and Sathish Gopalakrishnan (sathish@ece.ubc.ca)

Review and Challenges of Assumptions in Software Development
Md Abdullah Al Mamun and Jörgen Hansson (jorgen.hansson@chalmers.se)

A Criticality Decomposition Architecture to Integrate Encrypted Sensor Data in the Smart Grid
Dionisio de Niz and Lutz Wrage (dionisio@sei.cmu.edu)

3:00 - 3:30pm Workshop Ends. Coffee break

4:00 – 5:00pm RTSS@Work Demonstrations and Welcome Reception

Keynote Speaker

Dr. David A. Redman

AVSI
AEROSPACE VEHICLE SYSTEMS INSTITUTE

Industry Perspectives on Complex System Integration Issues

The Aerospace industry has encountered as aircraft integration is increasingly dominated by the cost of software as its complexity and size have grown exponentially. In an initiative called System Architecture Virtual Integration (SAVI) the major players in this industry are investigating technical solutions for use in next generation aircraft development. This talk discusses some of the key technical issues they are facing.

Biography:
Dave Redman is the Director of the Aerospace Vehicle Systems Institute (AVSI), a division of the Texas Engineering Experiment Station and part of the Texas A&M University System. AVSI is a global aerospace industry cooperative focused on addressing common issues through collaboration between industry, government, and academia. Dr. Redman joined AVSI in 2008 and has overseen the expansion of the organization to include European and South American Members, the first commercialization of AVSI technology, and the launch of several significant international projects.

Dr. Redman received a Bachelor of Science in Electrical Engineering from Michigan Technological University and a Masters and Ph.D. in Electrical Engineering from the University of Michigan. Following a postdoc at Sandia National Labs, Dr. Redman performed research in radiation effects in optoelectronics at the U.S. Air Force Philips Laboratory.

Prior to joining AVSI, Dr. Redman spent seven years at GE Aviation, holding various technology development positions. He led the development of a strategic road mapping process and other internal research and technology process. Dave also held the position of Chief Systems Engineer for Rotorcraft Products and served as the lead on a division-wide model-based engineering initiative. Part of this initiative was contributing to and guiding the GE participation on the AVSI SAVI project.

Abstracts:

1. FUSED: A Tool Integration Framework for Collaborative System Engineering

Mark Boddy, Martin Michalowski, August Schwerdfeger, Hazel Shackleton and Steve Vestel (martin.michalowski@adventiumenterprises.com)

FUSED is a tool integration framework that supports multiple engineers who are collaborating in the development of a diverse set of engineering models used for multiple purposes in multiple phases of development. FUSED is extensible to support a chosen set of modeling environments; a few examples from our work are requirements, solid geometry, computational fluid dynamics, dynamical systems, and vetronics/avionics. An extensible language approach is used, so that many FUSED capabilities are presented to domain experts as minor additions to familiar languages and tools. There is also a special FUSED language to specify compositions of models. Compositions may be used for multiple purposes, e.g., to specify multiple views of a component, verify inter-model consistency, specify part/whole assemblies, or apply design operations to models. One goal of FUSED is to reduce errors due to inconsistencies and emergent properties that occur across multiple models being developed by multiple domain-specific experts. For example, FUSED has an extensible typing and meta-typing system, and compositions may include powerful model verification environments. Another goal is improved support for concurrent, collaborative, mixed-initiative, evolutionary development processes. For example, FUSED was designed to support dependency tracking, change management and ripple effects analyses, version control and remote model server access, and mixed-initiative and multi-disciplinary collaborative optimization.

2. A Design Framework for Model-based Development of Complex Systems

Hristina Moneva, Roelof Hamberg and Teade Punter (hristina.moneva@esi.nl)

A Design Framework is presented that aims at capturing the design rationales in the process of designing embedded or cyber-physical systems. Its principal concepts cover storing the design rationales, which encompasses design decisions and analysis results, by linking design goals to concrete questions and analysis results for a particular scope of the system. The Design Framework does also provide a mechanism for using heterogeneous models for different system parts and linking them by means of essential design parameters and their dependencies. An elaborated conflict detection mechanism at different levels is provided in order to enable the designer to keep the design consistent throughout the process. The paper also presents first experiences in applying the prototype in industrial contexts.

3. Analytic Virtual Integration of Cyber-Physical Systems and AADL: Challenges, Threats and Opportunities

Jerome Hugues (jerome.hugues@isae.fr)

The design and implementation of cyber-physical systems gather multiple domains, from low-level physics up to complex control of systems to implement a full function. Such complexity requires particular strategy to characterize each level of abstractions, and then integration to ensure the system under consideration is correctly built. The advent of Model-Based Engineering is often perceived as a silver bullet to achieve all these complex tasks: the system designer can master its design through proper model artifacts (blocks, connections, properties, ...), virtual integration of system blocks, and analysis. However, current MBE processes usually cover vertical analysis, and address only a few

aspects like scheduling or behavioral analysis, while CPS would require also horizontal analysis of the system, combining analysis results. In this position paper, we review experiments on the use of AADL to design CPS, and highlight challenges, threats and opportunities to support analytical virtual integration.

4. From Abstract Component Descriptions to Timed I/O-Interfaces in AUTOSAR

Stefan Neumann and Sebastian Wätzoldt (stefan.neumann@hpi.uni-potsdam.de)

We present how based on a given AUTOSAR Software Component (SWC) a behavior model in form of a set of timed automata is derived. We show how abstraction is applied on the derived automaton model to create a timed interface that can be used for the purpose of compositional reasoning. When applying abstraction potentially information gets lost that is required for being able to distinguish between valid and invalid behavior. We introduce a methodical approach that allows finding out if the derived interface is expressive enough to exclude all invalid behavior of the SWC based on the derived interface. In the case that invalid behavior is not excluded by the interface, the process is able to generate example interactions with the SWC that lead to such an invalid behavior.

5. Quantitative Fault Propagation Analysis for Networked Cyber-Physical Systems

Linda Briesemeister, Grit Denker, Daniel Elenius, Ian Mason, Srivatsan Varadarajan, Brendan Hall, Devesh Bhatt, Gabor Madl and Wilfried Steiner (linda.briesemeister@sri.com)

This paper presents an approach to analyzing a model of networked cyber-physical systems for fault propagation. We present an implementation of a probabilistic logic model, which allows for reasoning via symbolic evaluation as well as numeric evaluation to perform a quantitative fault analysis. Our models are built from a few building blocks, which can be instantiated as standard or high integrity; communication paths can be made redundant, and finally, whole subsystem blocks can be replicated. We assume an underlying networking infrastructure of TTEthernet, which allows traffic of time-triggered, rate-constrained, or best-effort modes with different safety features. We apply our approach to a case study of a brake-by-wire system that contains communication flows with different traffic modes according to their criticality.

6. Integrating Sleep Scheduling and Compressed Sensing in Sensor Networks

Debojit Dhar and Sathish Gopalakrishnan (sathish@ece.ubc.ca)

We consider the interaction between the natural redundancy of sensor data for a variety of data gathering applications and the need for reducing energy consumption when such data is obtained through low-energy nodes communicating wirelessly. We employ compressed sensing as a mechanism for reducing the number of sensor nodes that need to be active at any given point in time for effective data reconstruction. Correspondingly we describe a decentralized coordination scheme that permits nodes to adjust their sleep and activity schedules subject to connectivity and coverage requirements that allow for satisfactory data reconstruction fidelity. Such problems are intractable and our schemes are approximation algorithms for minimizing the number of active nodes while adhering to the mentioned requirements. Our research contributions are two-fold: (i) we derive bounds on the performance of our coordination scheme and (ii) we evaluate the effectiveness of integrating compressed sensing with our

coordination scheme through simulations based on a data generation tool that models some physical phenomena.

7. Review and Challenges of Assumptions in Software Development

Md Abdullah Al Mamun and Jörgen Hansson (jorgen.hansson@chalmers.se)

The problems of implicit and invalid assumptions have been identified as one of the key reasons to project and software failures. Assumptions are available in almost all aspects of the software development from human factors to different software development activities. They also have influence on software quality attributes. The aim of this article is to provide a review of the existing work in assumptions management and find out the assumptions related challenges that should be mitigated in order to build better systems. The results show that assumptions are concerned with many different areas of software engineering and the existing approaches suffer from the lack of scope of assumptions categories and some concerns that are impacted by the assumptions. We believe a holistic assumptions management approach can mitigate assumptions related challenges by integrating concerned areas and contribute to build systems with smooth software integration and evolution.

8. A Criticality Decomposition Architecture to Integrate Encrypted Sensor Data in the Smart Grid

Dionisio de Niz and Lutz Wrage (dionisio@sei.cmu.edu)

In this paper we discuss the challenges that the integration of encryption protocols can impose on the scheduling of real-time systems in the smart grid. In order to address these challenges, we present a new task model and scheduling that takes advantage of the predictable bimodal nature of the execution of the tasks in the system. This nature allows us to use stream ciphers to decompose the computation of the encryption into two parts: a key stream pre-computation and a fast encryption computation using the key stream bits. This structure takes advantage of the fact that the trailing part of our bimodal task is not always executed (e.g., the actuation). When this happens we execute the keystream pre-computation to calculate and save values (key bits) to be used by the trailing part of the task in a future activation. We call this scheme computation buffering. The values produced during computation buffering reduce the execution time needed in the trailing part. Then we show how this scheme reduces both the utilization of the task (and the taskset) and the response time of the trailing part of the task. We then present the mapping of this task model to the multi-frame scheduling model and the modifications to the response time calculation. Throughout our discussion we use an example from the smart grid to illustrate both the challenges and the benefits of our solution.

FUSED: A Tool Integration Framework for Collaborative System Engineering

Mark Boddy, Martin Michalowski, August Schwerdfeger, Hazel Shackleton, and Steve Vestal
Adventium Enterprises
Minneapolis, MN, USA
{*mark.boddy,martin.michalowski,august.schwerdfeger,hazel.shackleton,steve.vestal*}*@adventiumenterprises.com*

***Abstract*—FUSED is a tool integration framework that supports multiple engineers who are collaborating in the development of a diverse set of engineering models used for multiple purposes in multiple phases of development. FUSED is extensible to support a chosen set of modeling environments; a few examples from our work are requirements, solid geometry, computational fluid dynamics, dynamical systems, and vetronics/avionics. An extensible language approach is used, so that many FUSED capabilities are presented to domain experts as minor additions to familiar languages and tools. There is also a special FUSED language to specify compositions of models. Compositions may be used for multiple purposes, *e.g.*, to specify multiple views of a component, verify inter-model consistency, specify part/whole assemblies, or apply design operations to models. One goal of FUSED is to reduce errors due to inconsistencies and emergent properties that occur across multiple models being developed by multiple domain-specific experts. For example, FUSED has an extensible typing and meta-typing system, and compositions may include powerful model verification environments. Another goal is improved support for concurrent, collaborative, mixed-initiative, evolutionary development processes. For example, FUSED was designed to support dependency tracking, change management and ripple effects analyses, version control and remote model server access, and mixed-initiative and multi-disciplinary collaborative optimization.**

***Keywords*-tool integration; system engineering; collaborative development; extensible languages;**

I. Introduction

We use the term "modeling environment" to mean a set of languages used to model systems from particular viewpoints plus a set of tools that automate associated engineering tasks. Some examples (all freely available) are OpenModelica for dynamical systems, BRL-CAD or Google SketchUp for solid modeling, Athena Vortex Lattice for computational fluid dynamics, TOPCASED for SysML requirements and AADL vetronics/avionics models. A modeling environment typically has a primary language for human entry and a number of additional data representations. The primary language could be a standardized textual language specified by a formal grammar, but it may also be specified informally by a GUI or API. Environments also include other representations, such as those generated by various model analysis and simulation tools. Our tool integration framework, which we call FUSED, has been designed so that it can be extended to support any given set of modeling environments.

The FUSED framework provides additional capabilities to specify compositions of diverse models, to script operations on these compositions, and to support dependency tracking and change management in a collaborative development environment. Examples include a composition of solid and dynamical models to capture different viewpoints of a component; a composition of components into an assembly or architecture; or a composition of a design optimizer with a design.

Developers must be able to specify how these new capabilities are to be applied. Our approach is to leverage existing modeling environments as much as possible by applying extensible language concepts. New FUSED capabilities are provided to domain experts as minor extensions to familiar languages. There is an additional, new FUSED language for specifying compositions of models developed in other modeling environments. This language deals with issues that are inherently multi-model and multi-environment and is primarily intended for system rather than domain-specific engineering.

Our concept of model includes traditional design models such as geometry models of aircraft wings and DAE models of vehicle dynamics. It also encompasses what might be called design process models, which are models that operate on other models. Examples in this category are design optimizers and trade space visualizers, model checkers and verifiers, and uncertainty propagators and global sensitivity analyzers. FUSED compositions can assemble parts into assemblies; *e.g.*, four wheels and a framework into a chassis. They can also specify applications of process models to design models; *e.g.*, a composition of a design optimizer or trade space visualizer with another composition of design models.

Defining a semantics for a composition of models that themselves have diverse and incommensurate semantics is a

This work was supported by DARPA under the META program contract # FA8650-10-C-7076. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited. Submitted 2^{nd} Workshop on Analytic Virtual Integration of Cyber-Physical Systems.

challenge. For example, the extensive and intricate geometric semantics defined for a solid modeling environment are at best weakly reflected in logical languages found in vetronics and avionics environments. Our approach is to combine concepts found in classical type theory with concepts found in theories of abstraction. The semantics of FUSED compositions are the abstractions of types found in the different models, at a level of abstraction that is common across the models.

We assume FUSED will be used in a development process involving many developers who are concurrently modifying models from many modeling environments. These models may be used at different phases and exist at multiple levels of abstraction, *e.g.*, preliminary requirements through as-built verification and validation (V&V). We assume models may be stored in a mixture of version control systems or accessed using a remote model server.

To ensure its utility in this kind of development, the FUSED framework is extensible to support any selected set of modeling environments and any selected set of object types from those environments. For the design and implementation of the FUSED framework itself, there are a number of technologies from which to choose as a basis. The UML meta-language approach and its associated representations and tools provide an obvious example [8]. Semantic web (web 3.0) concepts and representations are another option. FUSED supports UML domain languages like SysML and uses XML for tool data exchange, and the framework is designed to support web/SOA-style interactions such as compositions involving remote model servers, but our primary implementation technologies are higher-order attribute grammars and extensible language methods; specifically, the Silver attribute grammar specification language and code generation tools [2]. Silver, although based around the paradigm of a parser generator, has the full power of a functional programming language; attributes on syntax-tree nodes in Silver may have values that are themselves full syntax trees, and there are also formalisms and analyses for the concise specification and seamless composition of sets of language extensions, both on the syntactic and semantic levels.

The following sections provide overviews and illustrative usage scenarios for problems addressed, model composition semantics, modeling environments and compositions, concurrent collaborative development, and the architecture and implementation approach for the FUSED framework itself.

II. Problems Addressed

Many errors occur due to inconsistencies between models developed in different environments, and to misunderstandings by engineers trained in different disciplines and focusing on different issues. Because all models are abstractions, they represent only aspects of the system, developed to deal with particular sets of issues for particular purposes. There are characteristics of a system (sometimes called emergent properties) that are really captured only in sets of models. We want to reduce errors that occur due to inconsistencies between models or due to failure to consider interactions between models. Section III discusses some FUSED features that address these problems and presents an illustrative usage scenario.

Development is a mixed-initiative process. Automated design tools are becoming increasingly common and powerful, but these must ultimately be applied by human developers and as such the human/machine interactions are becoming increasingly complex. FUSED makes it easy for developers to apply automated design aids to complex models to support complex, mixed-initiative development activities.

Additionally, development processes are becoming increasingly collaborative, concurrent, and distributed. The waterfall model is being supplanted by processes in which product lines undergo continuous evolution throughout their life cycle. Advanced multi-disciplinary system engineering frameworks need to be well-integrated with supporting processes like distributed revision control, collaborative model development, and change management. FUSED integrates with advanced process support tooling to create an overall environment in which developers can stand on each other's shoulders instead of each other's toes. Section V discusses some FUSED features that address these issues and presents an illustrative usage scenario.

III. Composition Semantics

Our definition of "semantics" is "a way to map a string of symbols to a structure defined in some field of mathematics or science." Examples would include mapping a STEP file to a structure in solid geometry or mapping a string in the Modelica language to a system of hybrid differential algebraic equations.

For a particular modeling environment, we have to live with what we are given. This means we have to deal with different modeling environment semantics mapping strings to different mathematical domains, using different methods to define these mappings with differing degrees of rigor, *etc.* For example, a solid modeling semantics maps models to solid geometric objects with semantic rules such as "no two distinct solid parts may overlap in three space." These concepts are absent in the Modelica language definition. However, it is possible to map a moment-of-inertia tensor from the solid geometry domain into the domain of differential algebraic equations. There are some semantics that are explicitly defined only in the solid modeling environment, such as the concept of inertia of a material object (which is arguably a scientific rather than a purely mathematical semantics). But there are some semantics common to both modeling environments, such as units and linear operations on vectors in three space.

When two models are composed in FUSED, a developer can specify that one model may *publish* (pub) an object that is used to satisfy a *subscribe* (sub) in another model. FUSED automates this process and uses strong type-checking as part of what might otherwise be a manual cut-and-paste operation. When an object is specified for publication, what is actually published is a string of symbols whose semantics are an abstraction that is common to both modeling environments. In the case of a moment-of-inertia tensor, this is a 3×3 matrix of floating point numbers with associated units. Using our concept of language extension, FUSED provides additional information to this pub/sub process. For example, we extend the Modelica language slightly to include the concept of a frame of reference, in which case FUSED provides more semantics and type checking than standard Modelica.

In any particular FUSED installation, a set of common abstract types are defined. This means there are abstraction relations from types of objects in the various supported modeling environments to and from these common abstract types, made concrete in tools that can extract objects from various modeling languages and convert them to and from a common representation. In principle, the semantics of a pub/sub relation in a FUSED specification involves a mapping to a mathematical domain common to the two modeling environments, plus an abstraction relation for each of the two modeling environments based in theories of abstraction that would typically be different for the two modeling environments. Arguments then need to be made that operations on a subscribing model are correct; *i.e.*, information lost in the abstraction process does not render model analysis results invalid. In current practice, we just write Silver specifications for the common representations and conversion tools. This is an area rife with opportunity for further theoretical research and improved rigor.

Figure 1 depicts a usage scenario and FUSED model composition specification that illustrate some of these concepts. In this scenario, the system engineer would like some assurance that the solid model of an aircraft is consistent with the avionics model. In particular, the engineer would like assurance that the logical hardware resources and connections in the avionics model are consistent with electronics boxes and cables in the solid model.

Using the FUSED composition language, the system engineer specifies that an abstract representation of the static structure of a model be published for both models. This is a graph in which nodes and edges may have one or more types specified, where the set of node and edge types is just a list of identifiers specified in each model. (It is typical to provide language extensions that allow model developers to specify additional typing information if the standard language mechanisms are not sufficient.) For example, in the published abstraction of a solid model, nodes represent parts and assemblies and edges represent containment and attachment relations.

What the system engineer wants is assurance that there is a subgraph isomorphism between the logical hardware elements and connections in the avionics model and parts in the solid model, where the types of corresponding nodes and edges satisfy a specified type compatibility relation (*e.g.*, logical processors and buses in the avionics model map to LRU and cable parts in the solid model). This property can be concisely specified in SMTLib, a standard language agreed upon by the Satisfiability Modulo Theories research community for which there are a number of tools. This specification subscribes to two abstract model structures, and the SMTLib modeling environment provides a "check satisfiability" operation. By periodically running this FUSED composition specification, the system engineer receives ongoing assurance that this consistency property is maintained between the two models as they evolve during development.

IV. Model Environments and Compositions

A model environment is a set of languages and a set of tools that support a particular engineering discipline. An example is OpenModelica [6], to which the primary human input language is standard Modelica, and whose toolset includes a compiler and a simulator/solver. Any representation that holds data of interest at the system engineering level is also a language that FUSED at least needs to be able to read. For example, simulation trajectories and linearizations contain data of interest.

A specific model or related set of models is stored in a folder or project that is associated with a particular modeling environment. For example, the concepts of a modeling environment and an Eclipse project type are effectively synonymous, and there is an Eclipse plugin to support the OpenModelica project type.

Extending FUSED to support a particular modeling environment involves the following activities.

- Identify types of model elements that should be made visible at the system engineering level (in FUSED composition specifications). Ideally, many of these will already exist in the FUSED common type system; otherwise, they must be added.
- Define extensions to the primary language to support FUSED capabilities. Two near-universal extensions are the ability to declare that elements of a model are to be published (*i.e.*, made visible for use in FUSED composition specifications) or subscribed (*i.e.*, provided to a model when it is used in the context of a particular FUSED composition specification). Other common extensions allow additional typing information to be declared or support parametric/configurable models.
- Identify operations that can be performed on the model using the available modeling environment tools. (Mixed initiative operations that require some interaction with

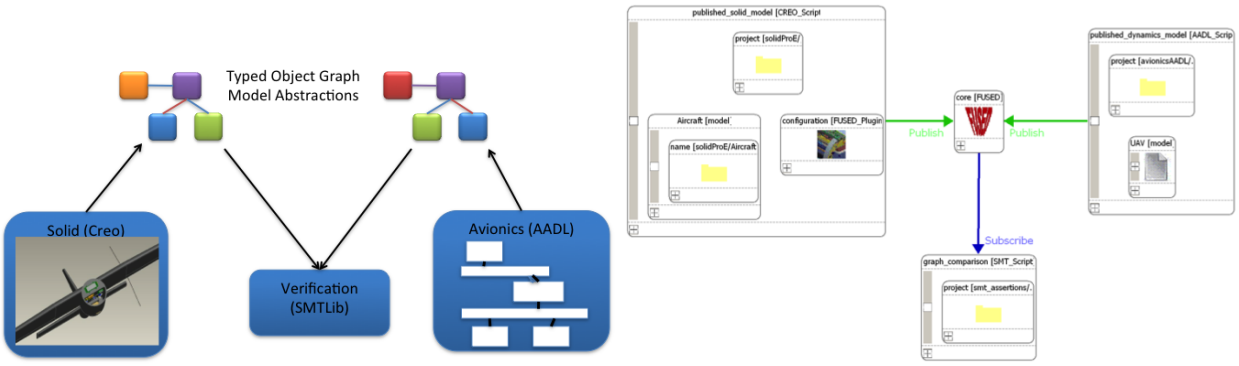


Figure 1. Model consistency usage scenario and FUSED composition specification

a domain expert should be permitted, although we have not demonstrated that to date.)

- Develop wrapper specifications in the FUSED language to present a set of convenient interfaces to higher-level FUSED programmers. This is not essential, but often makes the model more easily used by system engineers who are not familiar with either the model structure or the modeling environment.

The final section of this paper outlines how we implement these extensions. Basically, we do this by extending the existing build procedure to perform the identified operations automatically. Invocations of existing tools are wrapped with invocations of preprocessors and postprocessors that implement new FUSED capabilities and provide a proper interface into the FUSED framework.

When models are composed in a FUSED specification, that specification declares a pattern of publish and subscribe relationships between models together with other information such as scripting of operations to be performed. It would be quite tedious to declare this for every individual model element, and declaring a dependency for every individual element would also make composition specifications less robust to model edits. FUSED publishes structured sets of model elements, and FUSED subscribes are satisfied using a name space search procedure. The FUSED composition language includes a set of operations to perform restructurings, renamings, *etc.* (in the figures that show FUSED composition specifications, that is what the boxes labeled "FUSED" represent).

There is always a risk of mismatch, but this risk exists even if dependencies were manually specified for every individual scalar (which adds risk due to losing information about structural relationships between elements). FUSED mitigates this risk by providing a fairly powerful type system. In addition to basic types and type constructors like floats, integers, and arrays, FUSED supports the concept of a type qualifier. A type qualifier is additional information such as units, frames of reference, or uncertainties, that can also be associated with a model element. Modeling language extensions can be used to increase the amount of typing information provided and checked, and type qualifier operations are available in the FUSED composition specification language.

Figure 2 depicts a usage scenario and FUSED model composition specification that illustrate some of these concepts. In this scenario, three domain-specific models are developed to represent a UAV from three viewpoints. A solid model is used to capture and analyze the physical structure, a computational fluid dynamics model is used to analyze aerodynamic forces on the vehicle, and a dynamical systems model is used for flight dynamics. The models are configurable, so that system engineers can explore various combinations of design parameter choices.

The vehicle dynamics model needs data that can be obtained from a mass properties analysis of the physical structure, such as mass, moment-of-inertia tensor, and control surface areas. The vehicle dynamics model also needs a set of stability derivatives computed at a number of trim points, which can be obtained by CFD analysis. Rather than cut-and-paste for each UAV configuration, the vehicle dynamics model is modified slightly so that it subscribes to these values, which can be published by the other models. When the FUSED composition specification is executed, the models are configured, analyses executed, and data converted and copied as needed to obtain the specified vehicle dynamics analysis results; *e.g.*, a set of simulation traces. (This is similar in principle to what can be done with some existing commercial tools. A difference in this example is that FUSED supports complex object types with strong type checking.)

V. Development Processes

Models and associated assets are traditionally managed using revision control systems such as Subversion or Git. The design of FUSED also supports remote model servers. In order to execute a FUSED composition script, all that is needed is the ability to send a block of data to satisfy model subscriptions and receive a block of data published

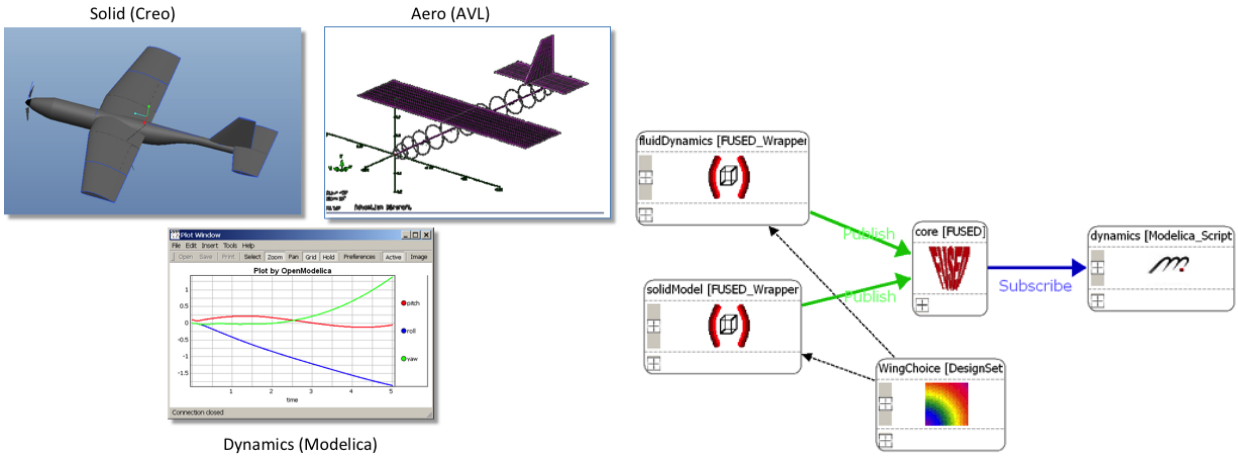


Figure 2. Model configuration and pub/sub scenario with FUSED composition specification

by a selected operation. To make full use of FUSED (*e.g.*, modeling language extensions), the model servers should also host a FUSED framework.

The set of all FUSED composition specifications in a development project captures much semantically rich information about the relationships and dependencies among all the models. Moreover, we conjecture that this information is captured much more concisely and manageably — the number of keystrokes needed to enter publish and subscribe declarations in models and abstract pub/sub dependencies in composition specifications is relatively small compared to the list of detailed element-to-element dependencies enumerated by the FUSED tools as it processes composition specifications. Moreover, abstract pub/sub relationships are robust with respect to changes in the individual models and thus more easily maintained. Finally, the fact that FUSED composition specifications are subjected to a variety of automated analyses provides some assurance of the correctness of these relationships.

The dependency relationships between models that are captured in FUSED composition specifications can be used in a variety of ways. Firstly, FUSED builds on traditional build/make dependency tracking technology. As FUSED composition specifications are executed, operations on individual models are only invoked as needed to provide up-to-date published data. Secondly, FUSED encourages and supports the use of configurable models. When an operation is performed on a model, this means the results depend on the values used to satisfy subscriptions in that model. We have prototyped results caching, so that repeats of an operation need not be done if the results of an earlier matching analysis have been cached.

Models are created for parts of the system in order to understand their properties from a particular viewpoint (*e.g.*, a Modelica model) to understand certain kinds of dynamical behaviors. We also want to support models that can be used as part of the design process itself, models that can be applied to other models. Examples of this are trade space models to support trade space exploration and Pareto frontier identification, various kinds of design optimization methods, and model and multi-model verification and validation activities. In cases like this, a FUSED composition is not an assembly of parts that is analyzed to determine properties of the assembly; it is executed to carry out a design activity on a model or composition of models. To support this, FUSED can publish and subscribe types of elements that are abstractions of models themselves; *e.g.*, the abovementioned typed object graph, as well as constraints and properties.

A variety of process models have been developed to perform uncertainty propagation and global sensitivity analysis to understand how model evaluation metrics depend on model design configuration parameters [3]. We are particularly interested in combining this sort of analysis with our dependency tracking capabilities to perform smart ripple effects analysis — is a change in a model big enough to significantly impact other models that relate to it in some way?

Figure 3 depicts a usage scenario and FUSED model composition specification that illustrate some of these concepts. In this scenario, the requirements engineering team is wrestling with the trade-offs between quality metrics like range, endurance, payload capacity, and cost. They are also still uncertain about what the available technology will make possible. What they would like to do is use a tool like Trade Space Visualizer [4] to explore the trade space and identify the Pareto frontier for the range of possibility given current technology.

The aeronautical engineering team has created an initial equational model (a spreadsheet) that allows these metrics to be estimated for a variety of design alternatives; *e.g.*, different choices of wing structure, batteries, motors, and propellers. If the aeronautical engineers knew the final re-

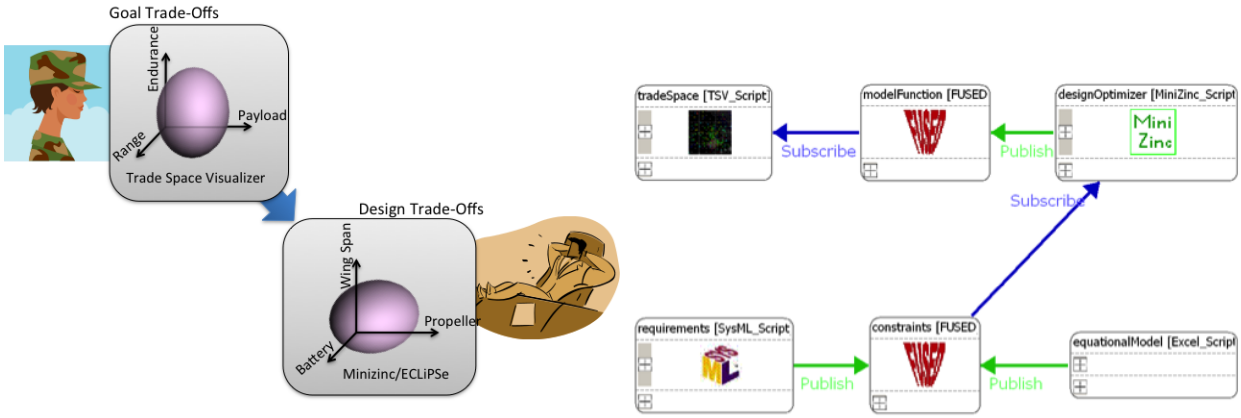


Figure 3. Mixed initiative design optimization scenario and FUSED composition specification

quirements for range, endurance, *etc.*, then they could select design choices that optimize the vehicle for the selected requirements. In fact, the aeronautical engineers constructed an optimization model (written in the widely-used MiniZinc language [5], for which a number of tools are available) that automatically makes good choices when given a final set of requirements. Such design decisions are irrelevant to the work of the requirements engineering team, who are merely interested in what could be achieved in terms of cost and endurance if they decided that low cost and high endurance are the dominant concerns of the end users.

To support these activities, the system engineering team creates a FUSED specification that composes the MiniZinc design optimization model with the requirements and equational models to create what is essentially a self-optimizing aircraft model that is parameterized by range, endurance, payload and cost — the parameters the requirements engineering team wants to explore. Trade Space Visualizer provides a variety of sampling methods that can be applied to an abstraction of a design model in which that model is a function that can be evaluated for a set of input parameters to determine the values of a set of evaluation metrics. In this FUSED composition specification, this is an abstraction that can be published by a model. The Trade Space Visualizer model subscribes to the functional abstraction that is published by the design optimization model. When this FUSED specification is executed, the Trade Space Visualizer will perform a statistical sampling of the design space by invoking the design optimization model/function, identify the points on the Pareto frontier, and provide a variety of graphical display formats that allow the requirements engineering team to explore this space.

VI. Extensible Framework

In this section, we briefly overview the current FUSED framework implementation architecture and technologies.

We currently use TOPCASED [7] for both its SysML and AADL modeling environments. TOPCASED is based on Eclipse, and one can think of an Eclipse project type as a modeling environment type. Figure 4 illustrates a set of model projects for a variety of modeling environments.

A central abstraction for a model is a set of operations that map a set of subscribed values to a set of published values. (This is an abstraction that can itself be published, *e.g.* for use by sampling-based uncertainty propagation models or gradient-search design optimization models.) These operations are implemented as a set of targets in a build script template that is created when FUSED is extended to support a particular modeling environment. The build scripts wrap calls to existing modeling tools with calls to preprocessors and postprocessors to handle language extensions, publish and subscribe operations, *etc.* Development of these preprocessors and postprocessors is another task performed when extending FUSED to support a particular modeling environment. There are currently cases where build scripts need to be tweaked manually to handle configurable models

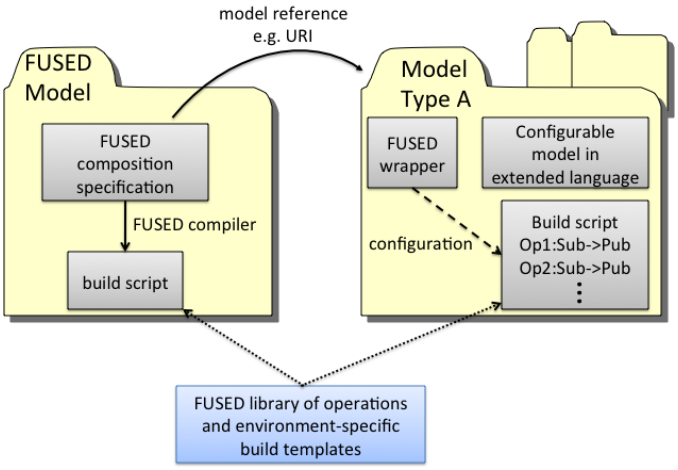


Figure 4. FUSED Project Types and Relationships

(acceptable to a software engineer but not to a system engineer), but eventually such configuration, when needed, should be generated automatically from a FUSED wrapper specification for a configurable model.

Specifications written in the FUSED composition language are models just like any other; *e.g.*, they can themselves be composed with other models. There is a modeling environment for the FUSED composition specification language itself. The build scripts here are not taken from a library, however; they are entirely generated from FUSED specifications by a FUSED compiler. When executed, these scripts mix calls to other model project builds with calls to various FUSED operations; *e.g.*, to perform restructurings on structured sets of published and subscribed model elements. Data is exchanged using an XML common type representation format, with conversions and type checking being performed by the preprocessors and postprocessors associated with each modeling environment.

The common types and their representations and conversion routines, and the preprocessors and postprocessors, are specified in the attribute grammar specification language, Silver. Attribute grammars are well-suited to concise specification of complex languages and data representation structures. Silver provides a rich typing system (*e.g.*, attributes on nodes of syntax trees may themselves be syntax trees, and generic attribute types are supported), so there is plenty of power to specify complex checks and conversions. Silver also has features supporting seamless language extension. This is chiefly done by specifying additional rules that recognize extensions to a language, and then placing "forwarding" constructs within those rules to specify a translation from the extended language back to the original. For example, a Silver-specified preprocessor would handle a subscribe declaration (an extension to the original language) by forwarding it to (rewriting it as) a declaration containing a literal value in the syntax of that language, after doing the appropriate look-ups and type-checks and conversions from a FUSED XML file. This re-writing is performed at the level of abstract syntax trees, rather than strings, allowing for more precise and fine-grained translations.

In summary, extending FUSED to support a specific modeling environment entails developing three kinds of new assets.

- Preprocessors and postprocessors for selected language representations, written in Silver. These implement language extensions such as publish and subscribe declarations.
- An ANT build template whose targets correspond to all the operations supported on models of that type. The template may contain subscriptions as needed to handle configurable models and parametric analyses.
- New common abstract types, including type-checking and conversion algorithms, may need to be specified in Silver and added to the FUSED common type system.

VII. Discussion

There is a trade-off involved in providing FUSED with a relatively powerful typing and language processing capability. This makes extending the framework more complex than if all it did was automate cut-and-paste of strings. On the other hand, it provides much more power. In our own demonstration exercises, we had unintended mismatches of units and frames of reference detected by our own framework. (Both of these classes of design defects are known to have resulted in a number of engineering failures and are still considered by many to be a source of significant risk.) More than this, though, is the potential for new capabilities like verification of non-trivial consistency conditions between models and support for more general mix-and-match of design process methods such as mixed-initiative multi-disciplinary design optimization. We conjecture that the potential benefits outweigh the additional extension complexity, which is a one-time overhead for each modeling environment in any case.

We believe that our choice of higher-order attribute grammar and language extension technologies was a good one, as they provide the added ability to compose models written in different languages by simply augmenting models slightly using simple syntax. We have not found it inconvenient to deal with UML languages via their XMI representations, and we have at least as much power to work at the meta-language and meta-type levels. Our language extension and pub/sub methods seem able to handle fairly complex model synchronization problems. The relationship to (semantic) web technologies is a bit more nuanced, and we currently tend to think of the relationship as one that invites synergistic integration. It is a combination that is needed to provide good support for distributed, concurrent, collaborative development. As one might expect, Silver is best suited for handling textual languages and representations having well-defined syntax, but relatively weaker for languages defined informally by the GUI or API of a specific tool.

Additionally, the overhead introduced by FUSED is minimal. While we have not conducted a thorough evaluation of the overall runtime impact, we observe that the time taken to translate FUSED-extended models to their equivalent base language is negligible and there is no impact on the simulation runtimes for these translated models when compared to the originals. Furthermore, while the process to extend FUSED to support a new modeling environment is in itself time consuming, it is a one-time effort. Assuming the software engineer develops a robust set of ANT build templates, their invocation will require no additional changes except for special cases. In these special circumstances, the software engineer only needs to modify the existing template rather than creating a new build script from scratch (a common process today).

FUSED is a work in progress. Our initial set of exten-

Table I
MODELING ENVIRONMENT EXTENSIONS

Category	Language	Toolset
requirements	SysML	TOPCASED
trade-off studies	(tool-specific)	Trade Space Visualizer
design optimization	MiniZinc	minizinc, ECLiPSe
spreadsheet	(tool-specific)	Excel
solid/geometric	(tool-specific)	Creo/ProE
fluid dynamics	(tool-specific)	Athena Vortex Lattice
dynamical systems	Modelica	OpenModelica
avionics/vetronics	AADL	TOPCASED/OSATE
model verification	SMTLib	Z3

sions supports nine domain-specific modeling environments (shown in Table I), but the set of language extensions and the set of common FUSED types is not what would be desired in a full system engineering project. The current tools for the FUSED model composition language only support publish and subscribe relations between models and combinations of operations provided by the associated environments, which supports only some of the various kinds of compositions discussed earlier. The FUSED implementation needs additional features — *e.g.*, occasional hand-configuration of build templates, acceptable to a software engineer but probably not to a system engineer — should be automated. As with all development projects at this phase, maturation is needed in the areas of documentation, error reporting, refactoring to facilitate easier extension, and general defect reduction.

Additionally, FUSED composition specifications capture much detailed information about dependencies between models, and we are developing ways that FUSED can use this data. We want to support specification and use of configurable models — component suppliers should provide a space of capabilities to system engineers who are making system-level trade-offs [1]. We want to support smart inter-model ripple effects analysis in which developers can analyze the magnitude and significance of changes or uncertainties in one model on other models. We want to support verification of consistency between models. We want to support highly parallel development processes, to the point that requirements engineering may be just another activity that goes on throughout a product and product line life cycle.

We have mentioned several areas where we believe that further research could yield significant benefits. Among these are an improved theory of semantics and abstraction, methods for smart change propagation analysis based on uncertainty and other "close enough" type qualifier information, experience with and improvement of inter-model consistency verification methods, improved caching of subscription-dependent model analyses, and more synergistic integration with advanced configuration management and model server and program management technologies.

VIII. ACKNOWLEDGMENTS

We would like to thank the Minnesota Extensible Language Tools group at the University of Minnesota for their Silver support, Lockheed-Martin for their aeronautical engineering and model development support, and the Defense Advanced Research Projects Agency (DARPA) for supporting this work under the META program contract # FA8650-10-C-7076.

REFERENCES

[1] Durward K. Sobek II, Allen C. Ward, and Jeffrey K. Liker, "Toyota's Principles of Set-Based Concurrent Engineering," *Sloan Management Review*, 40, 2, Winter 1999.

[2] Eric Van Wyk, Derek Bodin, Jimin Gao and Lijesh Krishnan, "Silver: an Extensible Attribute Grammar System," *Science of Computer Programming*, Elsevier Science, January 2010. See also http://www.melt.cs.umn.edu/.

[3] Christopher Hoyle, Irem Y. Tumer, Tolga Kurtoglu, and Wei Chen, "Multi-State Uncertainty Quantification for Verifying the Correctness of Complex System Designs," *Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Washington, DC, August 2011.

[4] Stump, G.M., Lego, S., Yukish, M., Simpson, T. W., Donndelinger, J. A., "Visual Steering Commands for Trade Space Exploration : User-Guided Sampling With Example," *Journal of Computing and Information Science in Engineering*, 2009. See also http://www.atsv.psu.edu/.

[5] Peter J. Stuckey, Ralph Becket, Sebastian Brand, Mark Brown, Thibaut Feydy, Julien Fischer, Maria Garcia de la Banda, Kim Marriott, and Mark Wallace, "The Evolving World of MiniZinc," *MODREF 2009*. See also http://www.g12.cs.mu.oz.au/minizinc/.

[6] OpenModelica, the Open Source Modelica Consortium, http://www.openmodelica.org/.

[7] TOPCASED, the TOPCASED consortium, http://www.topcased.org/.

[8] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, Vol 45, No 5, 2006.

A Design Framework for Model-based Development of Complex Systems

Hristina Moneva, Roelof Hamberg, Teade Punter
Embedded Systems Institute (Eindhoven, The Netherlands), e-mail: [firstname.lastname]@esi.nl

***Abstract*—A Design Framework is presented that aims at capturing the design rationales in the process of designing embedded or cyber-physical systems. Its principal concepts cover storing the design rationales, which encompasses design decisions and analysis results, by linking design goals to concrete questions and analysis results for a particular scope of the system. The Design Framework does also provide a mechanism for using heterogeneous models for different system parts and linking them by means of essential design parameters and their dependencies. An elaborated conflict detection mechanism at different levels is provided in order to enable the designer to keep the design consistent throughout the process. The paper also presents first experiences in applying the prototype in industrial contexts.**

Keywords: model-based engineering; modeling formalisms; multidisciplinary design; design framework;

I. INTRODUCTION

In the current practice of designing Cyber-Physical Systems (CPS), like MRI scanners and copiers, designers face the challenge of balancing the abilities to create a variety of models for analyzing system options and to track their role in the decision process. This often causes the following questions to pop-up in design processes:

- Why was a model made? Are the underlying assumptions still valid?
- Do the results still hold? Is the version of this model up-to-date?
- How do analysis results relate to design parameters?
- How is the system affected by a design parameter change?
- And many more questions…

The problem of capturing design rationales was already addressed in [1], [2]. We think that the problem is related to the challenges for CPS as observed in [3], i.e. 'Low productivity of CPS software engineers'. In this paper we present a so-called Design Framework that aims at capturing the design rationales in complex system design. The framework is developed by using our institute's knowledge about designing complex systems at mainly Dutch-based companies like Océ-Technologies, Philips Healthcare, and Vanderlande Industries.

CPSs do always require the involvement of specialists from multiple disciplines. The multidisciplinary character of complex systems does require different modeling approaches too. Figure 1 shows the multidisciplinary character of a generic design process. For design, a system overview is needed, which is one of the main concerns of system architecting. An architect compares the design options and takes decisions to choose from these which will impact the subsystem disciplines. The system overview directs the variety of disciplines, like the software, mechanical, and electronic hardware. Each of these disciplines, sometimes organized as departments, are bothered with delivering parts of the complete system design. System architecting is also fed by the disciplines themselves, e.g., when new design information within a discipline becomes available. Therefore a Design Framework that captures design rationales should capture overall system overviews as well as overviews within the disciplines to come up with adequate design information and their interactions.

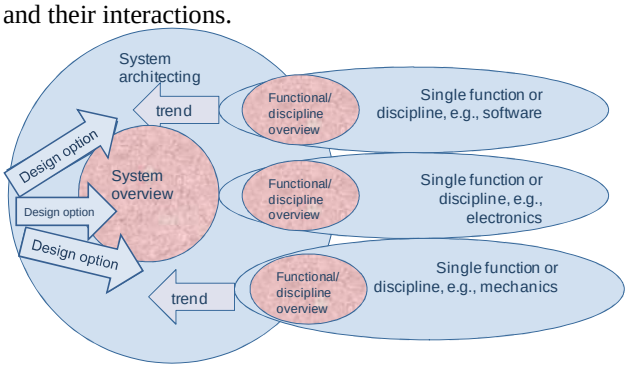


Figure 1 Multidisciplinarity in CPS design.

The models that are derived at the overall system level as well as in each of the disciplines are stored and should be related to each other. This is why the design rationale feature of the Design Framework is also denoted as *model and result management*. For relatively stable functions of the system the partial models are increasingly playing the role of starting point for implementation. The specification languages used for this are tailored towards the specific domain: they are called domain-specific languages.

A second motivation to develop the Design Framework is that today's complex system development requires *modeling* for analysis, see e.g., [4]. The main reason for modeling is rooted in the need to have design questions about a system answered, e.g., a performance model is made to answer a performance issue. We observe that multiple formalisms (the languages to make the models) are applied in industrial practice. The need for multiple modeling formalisms stems from their ability to answer different design questions. We also observe that CPSs are hardly modeled in a complete way [1]. Rather only the critical parts are modeled, because it is often not cost effective to model the complete product.

A third motivation for defining the Design Framework is the presence of *concurrent engineering* in design processes. All specialists from diverse teams are designing

the same system simultaneously and the communication between them is crucial for early error detection, which is usually hampered by implicit assumptions and failing communication of design decisions. Many organizations have organized their processes according to well-structured process approaches like RUP[1], DO-178B[2] and even may apply the CMMi[3] or ISO/IEC 15504 standards to assess their process maturity. On the other hand, we have observed organizations that develop complex systems, but do not have an explicit process in place at the detailed design level; in such cases only the main milestones are well defined. The Design Framework aims at supporting different implementations of processes. It therefore has a very simple and basic concept of a process step, i.e. the generic design step (see Section II).

The remainder of this paper first introduces the concepts of the Design Framework and its prototype tool in Section II. Next, experiences while applying the Design Framework are shown in Section III. Finally, an outlook for further development on the Design Framework is given and the paper is concluded in Sections IV and V, respectively.

II. DESIGN FRAMEWORK

In this section, we present the reasoning that led to the proposed approach, viz. a model that is targeting the support of a model-based design process as we believe it should be, i.e., with partial modeling in order to address the relevant design issues, as well as showing how it relates to current industrial practice. In the end, a prototype that embraces these concepts is presented.

A. The model

One way of classifying the variety of activities in a design process is according to their level of formality. On the one hand, more formal activities exist, which cover aspects like creating, manipulating, and analyzing concrete models. On the other hand, the more informal activities describe all the development steps that are made and the design decisions that are taken. These steps and decisions put all models in the context of the desired system. The informal activities can also be further distinguished: they are either activities related to the concrete design process or activities related to the design itself.

In order to support all design activities in a design process, a generic Design Framework model is proposed, see Figure 2. The Design Framework consists of three abstract levels or layers: design flow, design views, and models.

Each level incorporates the process and status aspects. The framework is designed to be generic and to be able to fit in any existent design process or development life cycle, any discipline or specific view, and to enable the usage of any formalism required in the working process. Due to the generality of the approach, multiple features can be added to support the designer solely based on the data, which is already present in the model – conflict detection, impact of a design decision, design exploration, etc. The ability to tailor this framework to the needs of a specific environment is considered an important usability issue. Some of these features will be introduced in Section IV.

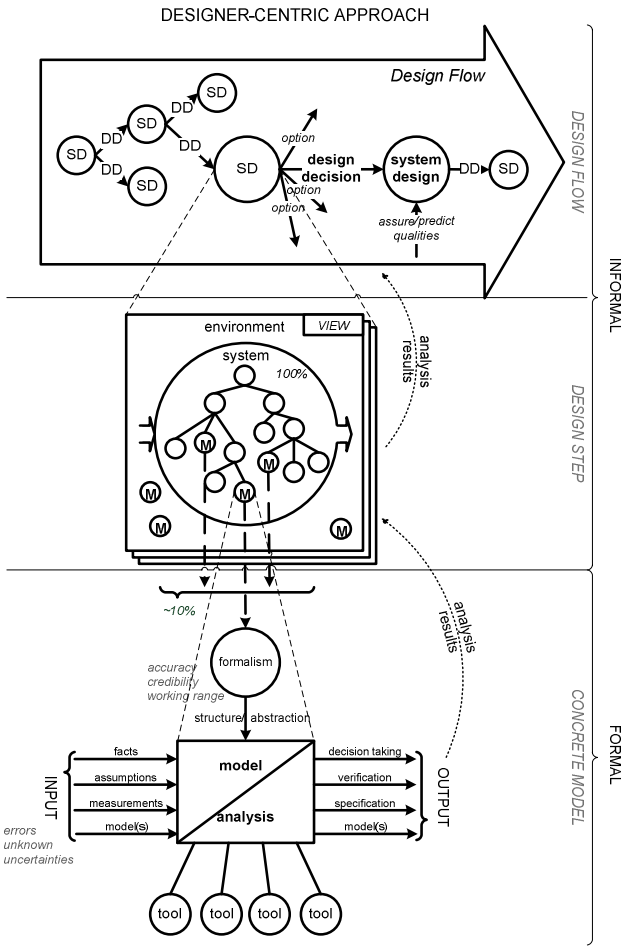


Figure 2. Design Framework model

1) The top layer: Design flow

Designing as an activity can be described by two main ingredients – taking a design decision, and, based on it, refining the designed system until the moment in time that a set of options enforces a next design decision to be taken (see Figure 3).

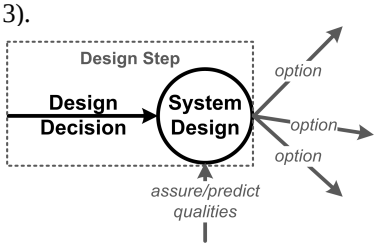


Figure 3 Design decision and design step

If we try to represent this process, it will be tree-like and comprise of a number of nodes (system designs) and

[1] http://www-01.ibm.com/software/awdtools/rup/
[2] http://www.esterel-technologies.com/do-178b/
[3] http://www.sei.cmu.edu/cmmi/

edges (design decisions or options) – see Figure 2, design flow level. Often, some design activities do not lead to the desired system (either dead branches/dead ends, or some possibilities are not explored further (yet)), alternatives, or the designers work on a few options in parallel in order to gain better in-depth insight in particular aspects of each option.

2) The middle layer: Design views

A design view may be considered as a specific representation of the system. It might represent a discipline, e.g., software, hardware, mechatronics, electronics, and material flow, or a specific aspect, e.g., performance, safety. Every view bears a unique decomposition of the system under design – see Figure 2, design view level. Structural basic blocks are used for the decomposition. They form the skeleton of the view. These blocks may contain or may be contained in other blocks.

The basic blocks are used as a container of the detailed models (see Figure 4). The blocks are formalism-independent and consider the model from a black box perspective. One block may contain multiple models, each of which is developed in order to analyze a specific concern or quality of the system or parts of it. If multiple models are present in one block, they are not considered as redundant, because they may model different aspects of the same system block. The necessity of being able to accommodate multiple models is also based on the need of answering multiple design questions by complementary analyses and the inability of any specific formalism to cover all required aspects. Note that multiple system aspects can be dealt with in one view (with multiple models) or in multiple views (each with its own model). It is up to the design team to decide which representation best fits their needs.

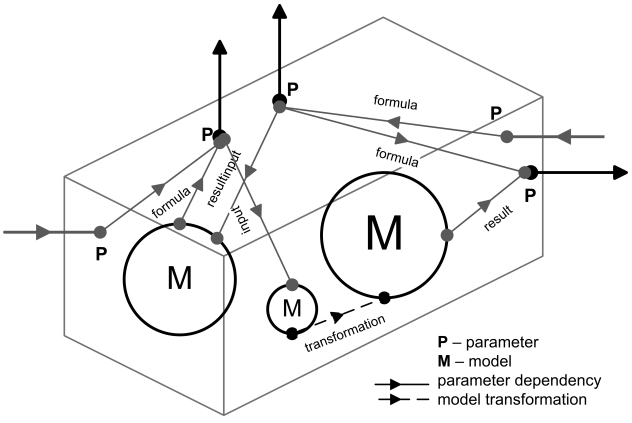


Figure 4 System block

Every block is characterized with a set of parameters. Each parameter might have a (range of) value(s) and a unit, and may have dependencies to other parameters. The parameters are used as a mechanism to couple blocks either within one or between multiple views. The interoperability of multiple formalisms and models is also enabled through these parameters. The basic block and the parameters that characterize it are used as an abstraction of the real detailed models. Due to the parameter dependencies, conflict detection between concrete parameters and their values – inputs or outputs of various models and their analysis results – becomes possible.

3) The base layer: Models

The need of modeling arises from the need of gaining in-depth knowledge of the system under design or its parts. Each model is expressed in its own formalism, which is suitable for analysis of specific aspects/questions. The model – see Figure 2, model level – usually has a number of inputs: a set of facts, assumptions, measurements, or even other models due to model transformations. The model, dependent on a chosen formalism and its degree of abstraction, having inputs with their errors, unknowns, and uncertainties, inevitably also has its own accuracy, credibility, and working range [5]. These are relevant attributes that should be known to the designer.

In general, multiple experiments can and will be performed on one model. The type of experiment is limited by the set of tools and their abilities. The decision to store the results of a particular experiment is under control of the designer. With any experiment, data should be stored about the tool that is used, its parameters, and the results. The results may be used for further decision taking, verification of assumptions or system qualities, or specification of some system parts. A specific result may be another model which in its turn is used for further analysis in case of model transformations. The latter allows for relating the design framework to so-called tool-chains.

4) Horizontal decomposition of the model

The vertical layering of the model can be naturally extended by a horizontal decomposition, see Figure 5. This decomposition is preserving the dual nature of design in itself: design process/activity versus design result/status, or in other words the 'why' and the 'what' aspects of the system under design. In the realm of this reasoning each horizontal layer can be interpreted as 'how' the activity can be performed or 'how' the design status can be detailed even more.

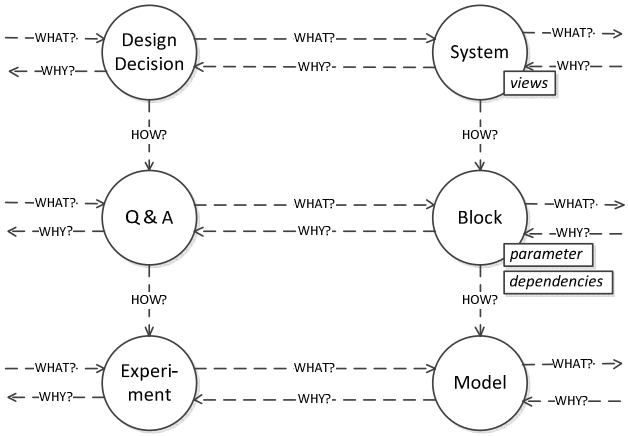


Figure 5 Design activities –'why & what & how' design reasoning

A typical design process involves taking design decisions, detailing the design, and checking whether or not the designed system meets all requirements. These

decisions result in a design status of the system design at this point in time (the 'what'), while the reason for the design status is the design decision itself (the 'why'). In order to understand the implication of a design decision, relevant questions should be posed and their answers evaluated (the 'how'). Usually such questions have a particular scope within the system – a particular system block. The relation between the system design and the system blocks is described by 'how' they are related to each other (the concrete system decomposition).

Similarly, system block characteristics are the reason for posing particular questions and vice versa. Often, in order to answer a design question, further modeling is required. The model can be analyzed in an experiment and its results will be supportive to providing answers. Hence, the experiment is seen as detailing the design question and the model as detailing the system block.

At all levels top-down and bottom-up reasoning is combined in reality. For example, the need for a particular experiment might predetermine the type of model to be built, but alternatively the model might constrain the type of experiment that can be done with it.

The explained design reasoning concerns just a single design step in the text above. In practice, multiple steps are required for the entire design process of a system: so many times this design reasoning will be applied.

5) Core domain knowledge

Most of the system developments are not a greenfield. That is, a lot of knowledge from other projects, from the system designers themselves, and common knowledge is already available from the start. Moreover, a known set of modeling tools is at hand for the developers to support them in their modeling activities. The presented design framework offers a means to store this information as well.

The main reason for including this in the framework is to be able to manage the interactions with changes in the core domain knowledge such as evolving knowledge resulting in adapted system patterns, and different versions of tools. This is a suitable way to store information about reusable library components as well.

B. The Design Framework prototype

These concepts were used as a starting point for developing a tool prototype. The prototype is based on a domain-specific language (DSL), also called meta-model, that represents the structure and relationships of all concept elements. This DSL was developed with the help of the Eclipse Modeling Framework[4], it being used for code generation in order to support rapid prototyping and early, continuous testing. Based on this meta-model also two graphical editors were developed, partially automatically generated by the Graphical Modeling Project[5].

The first editor is supporting the user when taking design decisions and storing the design rationales in the form of design questions and answers which are linked to the analysis results that support the conclusions. Moreover, the editor shows the current system design in term of views. The consecutive design steps result in a directed graph, where each node represents a design step. This editor is called the FLOW editor (see Figure 6). From each of the views, a second editor can be initiated. It is meant to represent the system structure and its parameters, dependencies, models, and experiments that belong to one view. This is the VIEW editor (see Figure 6).

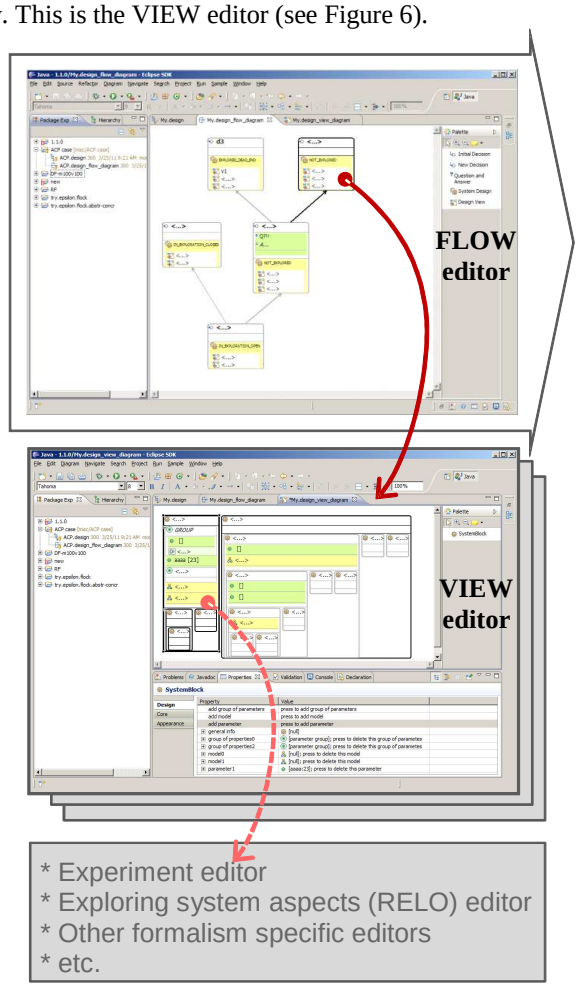


Figure 6 Design Framework– the basic set of graphical editors

Both editors allow to print the graphical representation which can facilitate various technical meetings by providing the most relevant and up-to-date status of the system under design.

The data of the Design Framework model is stored in a specific XML file, which serves as an input to all graphical editors. That allows easy sharing of a single "design" file among the development team and locally regenerating all graphical representations. It is recommended that along the design itself also all model files will be shared with the entire team.

Currently, the two graphical editors are part of the prototype, but a number of other possible editors are expected to become available (see Figure 6, bottom part). One such editor would support easier manipulation of

[4] http://www.eclipse.org/modeling/emf/
[5] http://www.eclipse.org/modeling/gmp/

model experiments – from more convenient way to store the data to possible automation of the execution. Another editor would support creating and storing exploration traces or diverse system aspects using the relationship-based exploration (RELO) approach, which is discussed in more detail in [6].

III. EXPERIENCES IN APPLYING THE DESIGN FRAMEWORK

Two industrial studies are presented to provide evidence that real industrial processes can be mapped to the concepts of the Design Framework. Moreover, the studies support one of our claims, i.e., that the design reasoning, one of the foundations of the Design Framework, can be observed in practice.

A. Architecture team discussions

In order to test the concepts as well as the prototype, a case study of an actual design process of one of our industrial partners was performed. The development of a new candidate for a customer system was already in place for a long period and still continues at the moment we are writing this paper. One of the authors participated in the weekly architect meetings and has observed the entire process in a period of 2-3 months. This served as an input for the Design Framework prototype and allowed us to gain experience with a real industrial problem at hand.

1) Case introduction

The result of the observations was the representation of the design process consisting of five design steps, see Figure 7. The system design – in terms of views, parameters, dependencies, models, and experiments –was getting enriched in each of these steps and was evolving over time. Each design step's goal was detailed in a number of design questions to be answered, which were addressing diverse system aspects and parts.

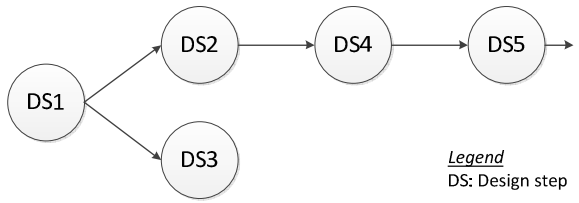


Figure 7 Abstraction of the design steps taken

At the moment when we joined the design process, the architectural team was discussing five possible candidates (DS1). As a consequence, in the first design step we had five separate views which represent these candidates. As these candidates share their basic equipment blocks with identical parameters, we also introduced a separate equipment view. A view with scenarios, which were used as test cases for these candidates, was added as well. These scenarios were based on customer characteristics and allow testing the suitability of the candidates for different market segments. Finally, the evaluation view was added, in which all important system KPIs are specified as well as their estimates per scenario and per candidate in order to compare the available options.

If we zoom in on a view, we see a graphical representation of the skeleton structure and an overview of all important artifacts of each block – parameters, their dependencies, models, and their analyses. For example in the case of candidate C4, the view has a system level block with only one additional level of decomposition into subsystems. Further decomposition at this point is not required, but can be added later. Its depth may vary at different places. At the system level three parameters are used to characterize some of the system aspects, where for example the cost per unit is dependent on the analysis results of two models. Also a number of models and their experiments are stored. The visual overview does only provide names of experiments, while all details – such as the experiment inputs and results, model path, tool, etc. – are accessible in the properties view of each block. The analyses results are used to formulate answers to the design questions of this current design step.

After studying the availability aspect of the candidates and their time-to-market, a number of candidates were discarded. This resulted into a new design step (DS2), where the number of candidates were limited, while other aspects of the remaining candidates were studied. In the meantime, one of the discarded candidates was still being evaluated as a possible solution for a customer by another set of people (DS3). It turned out that this candidate was very suitable to this type of customer and it should therefore be preserved. In order to deal with the different candidates, the architectural team decided for a more generic system, which could be configured by a number of design parameters into each of the three feasible candidate solutions.

This brought us to a next design step (DS4), in which the generic system's performance is examined. Meanwhile another aspect became dominant: a visualization of the systems in this domain was very important and a limited feasibility study was performed. Once the structure of the generic system was fixed, the design activities focused on the control aspects of the system (DS5), where other design concerns play a role. At this point our participation in the design process has ended.

Just to give an impression of the size of a typical design step in this study, the number of models used was 12 and they were referenced 21 times, 16 experiments, 85 system blocks for all views, 139 parameters, and 22 dependencies. The total number of design questions for all five steps was 13.

2) Observations

Based on the experience we gained while following the industrial design process, a number of observations are made. The first is that when using the Design Framework, it is easier to track the status of the design activities. For example, in the first design step, the architectural team intended to explore all possible options of candidates and scenarios and study their availability characteristics. For that purpose the evaluation view was created. When you

see its status in the VIEW editor, it is obvious that this intention was not carried out fully (see Figure 8).

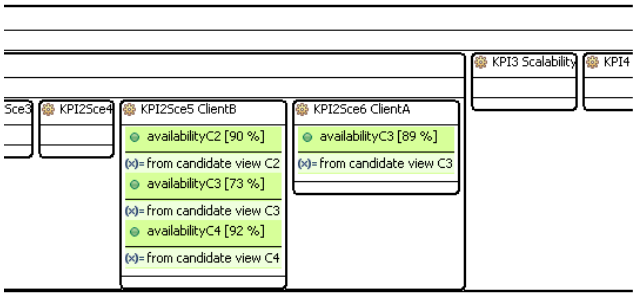


Figure 8 Evaluation view, only partially filled out

A second observation is the fact that some implicit design activities become more explicit when using the Design Framework. A good example is the fact that even though one of the candidates was discontinued, other people went on using this candidate in preparing a customer quotation. Just looking at the top level FLOW diagram, the status of the design and its top-level design decisions become evident, and it is easier to keep everyone up-to-date. In this respect, it is interesting that even less structured processes are fitting the proposed candidates well and can benefit from the proposed methodology to achieve better transparency and more explicit, stronger reasoning.

The third observation is that while following the design process, we observe also a pattern in the design activities, which is strongly related to our underlying design reasoning (see Figure 5). All design discussions were following the proposed pattern, but due to the fact that the architectural meetings were organized on a weekly basis there were a lot of activities that were happening in the background. The role of the meetings was to steer this process and take the main decisions.

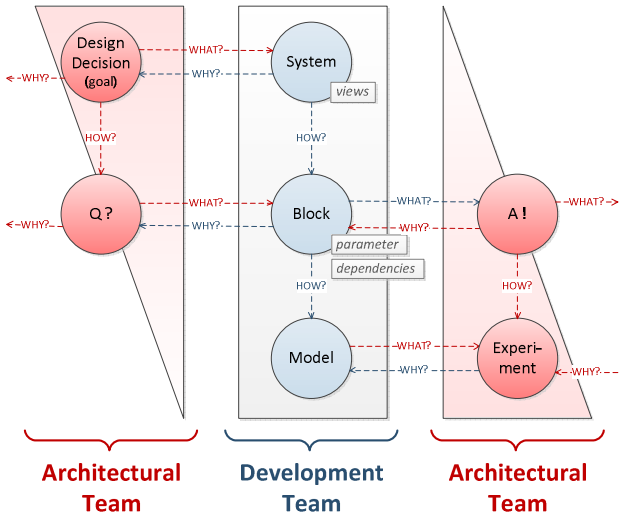


Figure 9 Elementary building block of the design process split according to responsibilities of architects and developers

A decision process starts with the formulation of a goal, based on the current status of the system under design (see Figure 9). This goal can be decomposed into a set of design questions, where each question has its own scope within the system. In order to answer these questions, a set of analyses will be defined, which also indicates the types of models that can be used to achieve these purposes. These activities belong to the architects' responsibilities. The development team is responsible for the actual development of the models and their analyses. These analyses will serve as preparation for the next architectural team meeting, where answer(s) based on the results can be formulated in an evidence-based manner.

Following the scope of the responsibilities of the architects, an architectural meeting has to focus on (see Figure 10):

1. Analyzing the experimental results (based on real measurements and/or models),
2. Answering the design questions from the previous meeting,
3. Identifying the next design goal and its questions, and
4. Defining the required models and experiments in order to prepare for the next meeting.

It can be imagined that the Design Framework tool can be used even during these meetings to structure them as well as to bring all up-to-date results into the room.

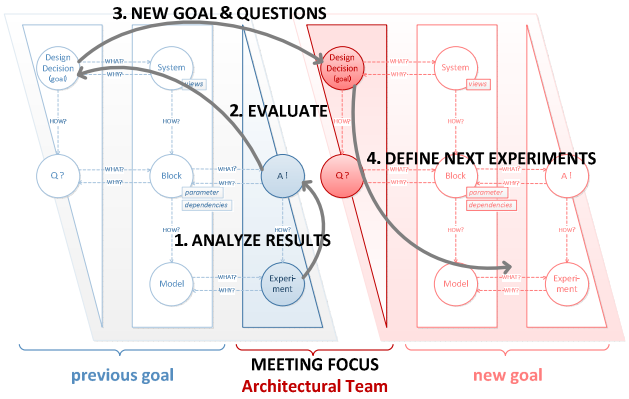


Figure 10 The focal steps of an architectural meeting

The fourth observation concerns the fact that the design process had to deal with a number of system candidates and all of those were simultaneously present in the system design process for some time. We choose to represent these candidates as separate views within the same design step. Another approach could have been to represent each candidate by a separate design step, together forming parallel branches in the design process. The approach we selected – representing each option as a view – has the advantage of allowing the candidates to share information available in the equipment view. If the second option had been selected – each candidate with its own design branch – the candidates could only have shared information via the core domain knowledge, which could also have been used to store the information about the equipment. Both approaches are feasible and equally suitable.

B. *Architecture overviews*

A second experience stems from a feasibility study in mapping an architecture overview upon the Design Framework. The architecture overview was defined to capture and to share architectural knowledge about the system latency aspects of one of the machines of one of our industrial partners. The overview was defined with help of the "A3 Architecture Overviews" method [2].

This method aims at capturing the information otherwise available in multiple and scattered documents and models as well as in people's minds. The method results into an A3-sized sheet of paper, where both sides are used – one with more textual and context information, the other with more graphical and model information. The overview has a few key elements, such as functional flow, physical view, system concerns, key parameters and requirements, design strategies, design decisions, assumptions, and known issues. A feasibility study was set up with the goal to follow the creation process of A3 overviews and to store the available design information in the Design Framework.

1) *Case introduction*

Creating A3 overviews is a time-consuming process that spans multiple iterations of creating its contents and communicating its status in order to validate it. Typically, in each iteration different design aspects are being questioned. The reason is that it is not simple how to select the most relevant information from various sources, which have different appraisals due to a range of perceptions and judgements. At the moment of writing this paper the overview creation process was still in progress.

The first version of the architecture overview was focused on the visual representation of the system functional flow and physical view along with a few of the key parameters and system constraints. The mapping to the Design Framework concepts was trivial. The functional flow and the physical view were represented as design views within the Design Framework. The flow decomposition was depicted by a hierarchy of system blocks. Each visual representation of some of the flow steps became a model attached to the corresponding system block. A similar mapping principle was employed for the physical view. The key parameters section contained a table with values, which was represented by parameters belonging to specific system blocks. The system constraints section was also translated into a set of parameters, but some of them also had dependencies to other parameters.

The successive versions of the A3 overviews contained more elaborated design information in terms of models. Some of the visual representations of flow steps were updated and more key parameters and system constraints were introduced. That resulted into a new design step in the Design Framework, where some models, parameters, and dependencies were updated and others were introduced. This version also had a partially filled text side of the overview, where some definitions of system parameters in different views were introduced as well as some system requirements and domain constraints. All these were also mapped to various models, parameters, and dependencies.

2) *Observations*

The main observation is the ease of mapping of the concepts of the A3 method and the Design Framework. While these two approaches have different goals, they are both trying to reveal and represent the design rationale of a system. That strengthens our confidence in the approach presented in this paper.

Another interesting observation is that while mapping the system constraints and design decisions there was design reasoning containing "because" in its phrasing. In order not to lose this design rationale, we mapped it to design questions and provided the reasoning as an answer to it. That allowed us making the design rationale even more explicit than just being hidden among multiple parameter values put in a textual form. Each design decision had its own goal of elaborating or understanding specific parts or aspects of the system in a better way. With the Design Framework it was possible to provide the design rationale in the form of questions and answers along its goal – not the entire set of concerns but just the relevant ones.

Knowing that the Design Framework and the A3 overviews can be easily mapped to each other, we think that the information stored in the Design Framework may be used for generating such architectural overviews. It is important to note that this type of overviews is meant to present just a limited but relevant set of information, while the initial data set may be much larger – coming directly from the architects themselves (the original approach) or from the Design Framework. At this point the challenge would be finding appropriate techniques 1) to select the information that is to be used and 2) to change its visual appearance to the best suited form for automatically created overviews.

IV. OUTLOOK

The two industrial cases for applying the Design Framework prototype show the applicability of the presented concepts and the reasoning framework behind them. All observations in terms of process and produced artifacts were seamlessly mapped onto the concepts.

However, the Design Framework is more than that. There are a number of features that provide even better support to its users [6]. These features can automatically be harvested to support design activities as a result of the generality of the presented concepts and the incremental actual data that is provided by the user in the course of designing. Here we list those features:

- Conflict detection. Based on the data that is being stored in the framework during designing, a continuous conflict detection mechanism can be employed to support the user. The mechanism applies to parameters, their values, and their dependencies and it will continuously check for incompatibilities. That also includes model versions and experiment inputs and results.

- Decision tree and impact of design decisions. An important feature is the ability to deduce the impact of each design decision on the system under design. That can be obtained by differencing any pair of design steps and observing the changes in the designs. This feature facilitates retrospection of the decisions taken and their exact impact on the system. Often, when taking a decision this does not imply that designers are aware of the changes required to the design in advance. In such cases retrospection may help to improve the understanding as well as to give space for improvements when necessary.
- Exploration or cause-effect analysis. This feature, also called relationship-based exploration (RELO), allows expanding interactively a different visualization, starting from any parameter and continuing the exploration in any direction based on the provided dependencies to other parameters. We can use this for showing and storing specific aspects of the system under design in cases where views reflect disciplines and the system qualities/concerns are spread among them (e.g. availability, performance).
- Other features concern different representations of design activities based on time or based on the stability of system parts, reuse of information in different forms, collaborative work and meta-processes, as well as tailoring the concepts and tool to custom domain needs.

The current and future activities target the completion and maturity of the abovementioned features, which will enable us to validate the full potential of the presented approach. While these efforts are still in progress and are based on the confidence we obtained while applying the current version of the Design Framework prototype on various industrial problems, we focus now on real-time application of our tool inside the industry.

Other activities will be focused on investigating the feasibility of this approach for prescriptive processes as well. Such repetitive processes can be found in some domains, such as aerospace and chemical industries, but also in different lifecycle phases, such as sales and certification. Another type of investigation that we envision is on applying the Design Framework in areas as design space exploration and set-based design. Last but not least, we would like to find out more about the different roles of people involved in design – sales, architects, developers, testers, integrators – and their primary interests and possible benefits when using our approach.

V. CONCLUSIONS

We have shown the concepts of a Design Framework that supports development in industrial contexts and have presented observations that show its feasibility and relevance. The added value of the Design Framework approach compared to the existing way-of-working of developing CPSs is:

1. Integrated support of design activities, covering three related, but different levels – process (design flow), system decomposition (views), and models and analyses.
2. Characteristic of the approach is its flexibility of using different tools and formalisms, which enables designers and architects to deal with model heterogeneity,
3. Design-time error detection by keeping the dependencies explicit between design decisions, system elements, and models,
4. Support for incomplete modeling, and
5. Ability to show the impact of each design decision on the system under design.

Points three and five are still to be validated in industrial environments.

The framework is domain independent and can be applied in a variety of industries when designing cyber-physical systems in a model-based way. Along all its advantages, there are also some concerns as well. The main difficulty of applying it in industry is the need of introducing a new way of working, which involves exercising more discipline and tidiness and may not be easily accepted. We expect, and are supported in this by our observations, that this will pay off with reduced time for communication, less implicit decisions, and a good overview to all members involved in the design process.

ACKNOWLEDGMENT

This work has been performed as part of the "Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems" (MULTIFORM) project, supported by the Seventh Research Framework Programme of the European Commission.

Grant agreement number: INFSO-ICT-224249.

REFERENCES

[1] W. Heemels, E. van de Waal, G. Muller, A multi-disciplinary and modelbased design methodology for high-tech systems, in: Conference on Systems Engineering Research (CSER 2006), Los Angeles, CA.

[2] G. Bonnema, P. Borches, Design with overview – how to survive in complex organizations, in: Proceedings of the International Council on Systems Engineering International Symposium (INCOSE 2008).

[3] K.H. Kim, Challenges and Future Directions of Cyber-Physical Software, in: 34th Annual IEEE Computer Software and Applications Conference (COMPSAC), pp.10-13, 2010. DOI: 10.1109/COMPSAC.2010.89.

[4] D. Schmidt, Cover feature – Model Driven Engineering. IEEE Computer, February 2006, pp.25-31.

[5] G. Muller, Coupling enterprise and technology by a compact and specific architecture overview, in: Proceeedings of the International Council on Systems Engineering International Symposium (INCOSE 2007), San Diego.

[6] H. Moneva, R. Hamberg, T. Punter, Are you aware of the design decisions? On how modeling should support design, In: Proceedings of 1st Workshop on Model-based Engineering for Embedded Systems Design (M-Bed 2010) at DATE 2010, Dresden, Germany, 12th of March 2010.

Analytic Virtual Integration of Cyber-Physical Systems & AADL: Challenges, Threats and Opportunities

Jérôme Hugues
Université de Toulouse, ISAE
10, Avenue E. Belin 31055 Toulouse Cedex 4, France
Email: jerome.hugues@isae.fr

***Abstract*—The design and implementation of cyber-physical systems gather multiple domains, from low-level physics up to complex control of systems to implement a full function. Such complexity requires particular strategy to characterize each level of abstractions, and then integration to ensure the system under consideration is correctly built. The advent of Model-Based Engineering is often perceived as a silver bullet to achieve all these complex tasks: the system designer can master its design through proper model artifacts (blocks, connections, properties, ...), virtual integration of system blocks, and analysis. However, current MBE processes usually cover vertical analysis, and address only a few aspects like scheduling or behavioral analysis, while CPS would require also horizontal analysis of the system, combining analysis results.**

In this position paper, we review experiments on the use of AADL to design CPS, and highlight challenges, threats and opportunities to support analytical virtual integration.

I. Introduction

The design and implementation of cyber-physical systems gather multiple domains, from low-level physics up to complex control of systems to implement a full function. Such complexity requires particular strategy to characterize each level of abstractions, and then integration to ensure the system under consideration is correctly built. The advent of Model-Based Engineering is often perceived as a silver bullet to achieve all these complex tasks: the system designer can master its design through proper model artifacts (blocks, connections, properties, ...), virtual integration of system blocks, and analysis.

However, current MBE processes do not cover enough analysis, and address only a few aspects like scheduling or behavioral analysis. Still, we note there is a strong intrication of concerns when dealing with analysis of Cyber-Physical Systems: physics impose its pace on the system mechanical behavior or electric state; which impose constraints on signal integrity (value, capture) and energy; which in turns impact the hardware platform and thus the software running on it. The opposite flow also exists: to implement a particular control function, software imposes constraints on timing, memory that are to be fulfilled by the underlying system and environment, and their validation/verification.

In this paper, we review current experiments on the use of AADL to design CPS, and highlight challenges, threats and opportunities to support analytical system composition. Our position is built from multiple experiments conducted with academic and industrial partners from the last 5 years.

In section II, we briefly introduce Cyber-Physical systems, listing known issues and building blocks for the design of CPS. In section III, we introduce the AADL language and show it is a convenient support for expressing the architecture of a CPS. In section IV, we provide a comprehensive lists of initiatives to provide support for the engineering of CPS using the AADL. In section V, We list several challenges for the applicability of Model-Based Engineering for the design of Cyber-Physical Systems, based on experiments around AADL.

II. From embedded to cyber-physical Systems

Cyber-Physical Systems (CPS) has been recently defined as an extension to typical embedded systems. In the latter, the emphasis was on the computational elements, with focus on resource constraints, timing, safety, etc. CPS widen the scope of concerns to include physical elements, such as the nature of input/output signals and their effect to the environment, and more importantly the cooperation of computing and physical elements, potentially through networks. This is a significant paradigm shift as it forces tighter cooperation between separate domains and disciplines.

In this section, we review specificities of CPS, and place them to typical engineering practice.

The rise of CPS: The integration of physics is not a new concern, it was also part of most if not all embedded systems, e.g. control function for planes, automotive electronics, home appliances, etc. From a computer-perspective, physics is hidden through the discretization of inputs, command laws and discrete events. Conversily, the computing power is hidden behind a set of analogue or digital inputs. Interactions are limited to a precise set of inputs ranges, values and timing from well-defined actors.

In [25], the author claims that the key difference between CPS and classical embedded systems come from their networked nature, and the new interactions they propose. Such change in architecting computer-based systems is actually pervasive in most domains where a processor is used: consumer electronics, medicine, aeronautics and

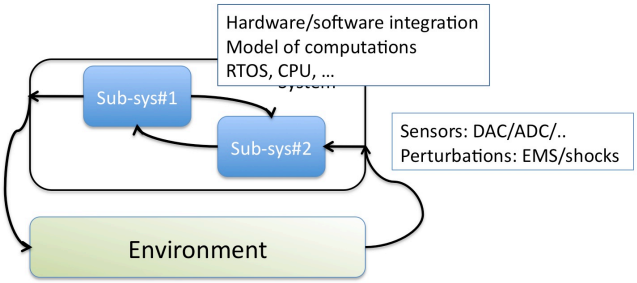


Figure 1. System breakdown structure

space domains. Embedded systems moved from basic I/O manipulation function (e.g. brake-by-wire) to more complex ones such as aggregating information from sensor networks on a car to optimize fuel consumption. To some extent, CPS have the potential to change many aspects of computing [33].

Integration issues in CPS: Actually, it is the *integration* of advanced services that makes CPS disruptive. In the same time, the *correct* integration of new services is what could limit them. This integration is many-fold, and follow the system breakdown structure: sub-systems interacting inside the system, and the system interacting with its environment (figure 1) :

- system/environment integration:
  - how to carefully manage inputs/outputs? what are the potential cause of failures/hazards and how to mitigate them? This covers many dimensions: electromagnetic compatibility, signal conditioning, human factors, malicious or adverse behaviors, ...
  - what are the timing constraints of the system? and how to accommodate systems with multiple independent discrete clocks?
- system/sub-system integration:
  - how to define the boundary of the system blocks? their interface? their contract in terms of service required/provided? of non-functional properties?
  - what are the characteristics of the hardware platform? the model of computation of the programming language? the constraints of the Real-Time Operating Systems?
  - how to verify, validate, test or qualify sub-systems?
  - what is the required granularity to ensure the correct integration of each block?
- properties determination:
  - how to express the properties of the system? what are the relevant metrics? how to extract information from the data sheet of basic blocks like CPU or network interfaces? what is the relevant level of abstraction for a particular analysis? what to model in a particular formalism and when to switch to a particular analysis tools?
  - how to evaluate the properties how the system? what are the cross-dependencies accross layers? (e.g. impact of energy scaling on CPU performances? impact of CPU performances on enerygy? ...). These dependencies shall be tracked so that once design choices are locked, other parameters are updated accordingly;

This list cannot be exhaustive, yet it is illustrative of the many challenges that arise when designing a CPS. Several workshops gathering the academic and industrial communities were held to discuss these challenges. This has been accompanied by a number of projects to study them.

Models as building blocks for CPS engineering: The agenda for these projects focus on the many required analysis: performance analysis (scheduling, network analysis, ...), memory and processor (latency, jitter, issues with cache and pipelines), programming languages (simpler, smarter, notion of model of computation), formal methods (breaking limits in scalability of model checking, complexity of logic formula, notion of time, probability, etc.), etc. Each analysis rely on a particular abstraction of the system: a model to be manipulated electronically for better efficiency.

Hence, model-based engineering (MBE) emerged as a convenient way to build models of systems to ease their analysis. Several tools have been developed, ECLIPSE being now the dominant platform, supporting UML and its companion profiles MARTE [30] and SysML [29] , and the AADL language [36]. Proprietary tools like SCADE Studio or Matlab/Simulink are also available. Each tool supports different formalisms to express a system, and transformation engines to perform a wide range of analysis (such as behavioral, timing, safety) and eventually code generation.

The capability to define models and analyze them pave the way to virtual integration of subsystems and their analysis: a descriptive model of the system is built; the level of details of each block, and their interconnection is expressive enough to perform a complete analysis prior to actually build it. This allows for early trade-off analysis and detection of defects in the specifications, functional implementation or non-functional properties. Such analytic capabilities build upon existing model processing capabilities, typical analysis techniques but also requires new innovative frameworks to address new level of complexities in design. Yet, one can question whether CPS and MDE are a good match: the level of maturity of tools, and its inherent complexity poses several issues to the industrial community. At the same time, many research projects show that some benefits are at hand.

In the following, we review challenges, threats and opportunities in the design of CPS from a MDE perspective. We take as an illustrative example ongoing works performed around the AADL language. This choice is driven by the author's experience, yet most conclusions apply equally to other modeling languages. We present several areas where challenges arise: system engineering, applicability of ana-

lytic frameworks, case studies and education.

III. Overview of the AADL

The "Architecture Analysis and Design Language" AADL is a textual and graphical language for model-based engineering of embedded real-time systems. It has been published as an SAE Standard AS-5506A [36]. AADL is used to design and analyze software and hardware architectures of embedded real-time systems.

The AADL allows for the description of both software and hardware parts of a system. It focuses on the definition of clear block interfaces, and separates the implementations from these interfaces. It can be expressed using both a graphical or a textual syntax. From the description of these blocks, one can build an assembly of blocks that represent the full system. To take into account the multiple way to connect components, the AADL defines different connection patterns: subcomponent, connection, binding.

An AADL model can incorporate non-architectural elements: embedded or real-time characteristics of the components (execution time, memory footprint, ...), behavioral descriptions. Hence it is possible to use AADL as a backbone to describe all the aspects of a system. Let us review all these elements:

An AADL description is made of *components*. The AADL standard defines software components (`data`, `thread`, `thread group`, `subprogram`, `process`) and execution platform components (`memory`, `bus`, `processor`, `device`, `virtual processor`, `virtual bus`) and hybrid components (`system`).

Each Component category describe well identified elements of the actual architecture, using the same vocabulary of system or software engineering:

- *Subprograms* model procedures like in C or Ada. *Threads* model the active part of an application (such as POSIX threads). AADL threads may have multiple operational modes. Each mode may describe a different behaviour and property values for the thread. *Processes* are memory spaces that contain the *threads*. *Thread groups* are used to create a hierarchy among threads.
- *Processors* model micro-processors and a minimal operating system (mainly a scheduler). *Memories* model hard disks, RAMs, *buses* model all kinds of networks, wires, *devices* model sensors, ...
- *Virtual bus* and *Virtual processor* models "virtual" hardware components. A virtual bus is a communication channel on top of a physical bus (e.g. TCP/IP over Ethernet); a virtual processor denotes a dedicated scheduling domain inside a processor (e.g. an ARINC653 partition running on a processor).
- Unlike other components, *Systems* do not represent anything concrete; they combine building blocks to help structure the description as a set of nested components. *Packages* add the notion of namespaces to help structuring the models. *Abstracts* model partially defined components, to be refined during the modeling process.

Component declarations have to be instantiated into subcomponents of other components in order to model an architecture. At the top-level, a system contains all the component instances. Most components can have subcomponents, so that an AADL description is hierarchical. A complete AADL description must provide a top-most level system that will contain certain kind of components (*processor*, *process*, *bus*, *device*, *abstract* and *memory*), thus providing the root of the architecture tree. The architecture in itself is the instantiation of this system, which is called the *root system*.

The interface of a component is called *component type*. It provides *features* (e.g. communication ports). Components communicate one with another by *connecting* their *features*. To a given component type correspond zero or several implementations. Each of them describe the internals of the components: subcomponents, connections between those subcomponents, ...

An implementation of a thread or a subprogram can specify *call sequences* to other subprograms, thus describing the execution flows in the architecture. Since there can be different implementations of a given component type, it is possible to select the actual components to put into the architecture, without having to change the other components, thus providing a convenient approach to configure applications.

The AADL defines the notion of *properties* that can be attached to most elements (components, connections, features, ...). Properties are typed attributes that specify constraints or characteristics that apply to the elements of the architecture: clock frequency of a processor, execution time of a thread, bandwidth of a bus, ... Some standard properties are defined, e.g. for timing aspects; but it is possible to define new properties for different analysis (e.g. to define particular security policies).

AADL is a language, with different representations. A *textual* representation provides a comprehensive view of all details of a system, and *graphical* if one want to hide some details, and allow for a quick navigation in multiple dimensions. In the following, we illustrate both notations. Let us note that AADL can also be expressed as a UML model following the MARTE profile [13].

The concepts behind AADL are those typical to the construction of embedded systems, following a component-based approach: blocks with clear interfaces and properties are defined, and compose to form the complete system. Besides, the language is defined by a companion standard document that documents legality rules for component assemblies, its static and execution semantics.

The figure 2 illustrates a complete space system, used as a demonstrator during the ASSERT project. It illustrates how software and hardware concerns can be separately developed and then combined in a complete model.

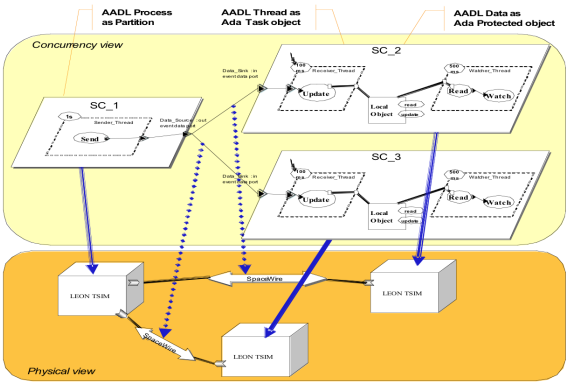


Figure 2. IST-ASSERT demonstrator

As we mentioned earlier, AADL, or other like MARTE or EAST-ADL provides similar constructs, and are conceptually really closed as underlined in [22]. A natural question is thus to review missing blocks for Cyber-Physical Systems Engineering. We discuss this point in the next section.

IV. AADL for engineering CPS

AADL provides interesting features to model Cyber-Physical Systems, analyze them but also implement them. In this section, we show that there is currently a good coverage of support tools to assist CPS designers. Actually, many tools provide support for AADL[1]:

- *Modeling:* TOPCASED [12], OSATE [37], and Stood [11] provide AADL modeling features for both textual and graphical variants;
- *Model of computation and architectural patterns:* AADLv2 annexes define patterns for supporting IMA architectures; other initiatives provides patterns for the Ravenscar computational model [18] or synchronous languages [42], [28];
- *Scheduling analysis:* the Fremont toolset [40] and Cheddar implement AADL performance analysis methods [38]. Gateways from AADL to the Cheddar and MAST tools are also available in TASTE;
- *Dependability assessment;* AADL provides an annex for modeling propagation of error, to be updated for AADLv2. Besides, connection with verification tools has been experimented for instance in the COMPASS project [5], the ADAPT toolset [35] and RT-Edge [20];
- *Security:* Patterns have been defined to model MILS security patterns [10], [19];
- *Model optimization:* optimization can occur across several dimensions: number of processors[16], use of communication buffers [15], allocation of threads to processors [8];
- *Behavioral analysis*: mapping to formal methods and associated model checkers have been defined for Petri Nets [34]; BIP [32], [6]; FIACRE [3]; RT-Maude [28];
- *Performance analysis:* performance of the system can be evaluated either at the level of generated source code [17], or from the description of interactions and I/Os in the system [27];
- *Code generation:* Ocarina implements Ada and C code generators for distributed systems [24], a mapping for RTSJ has been defined in [4]; AADS completes the range of language to add the hardware description language System-C [41]. Other initiatives exist to map AADL to synchronous languages like SIGNAL [26] or Lustre [21].

Several projects build on the foundations of these AADL tools to build integrated toolsets: the TASTE toolset driven by the European Space Agency [7]; the "System Architecture Virtual Integration" (SAVI) by the Aerospace Vehicle Systems Institute [14] an initiative gathering numerous key players from the aeronautics domain, and MASIW developed by the ISPRAS in Russia [23]. These integrated toolsets have to face many challenges, like the integration of additional modeling notations like SysML [2], or SCADE and Simulink [9].

Hence, after more than 10 years of development around the AADL, and the seminal paper from [1], one can assert that AADL provides a complete toolbox for designing Cyber-Physical Systems. Yet, we claim this is partially true, we develop arguments for challenges, threats and opportunities to support analytic virtual integration of CPS using the AADL in the next section.

V. Challenges, threats and opportunities to apply AADL to the design of CPS

In the previous section, we presented tools, methods and processes that support analysis of AADL models. Each initiative provide a partial solution to evaluate design correctness from one particular point of view. However, CPS require the combination of several aspects. This poses many challenges for the use of AADL for CPS.

A. Heterogeneous models as inputs

Although AADL is perceived by many partners as a potential solution to represent the system at some point, it is not the only artifact: one need complementary models to represent the physical environments (e.g. using Modelica), but also to design the system itself. For instance, using SysML for modeling high-level requirements, SCADE or Simulink for functional modeling. This is a strong requirement induced by the heterogeneity of domains, mixing system engineering, software/hardware and control/command concerns that span across the system lifecycle.

In this context, the challenge is two-fold

[1]An updated list of supporting tools, projects and papers can be found on the official AADL web site http://www.aadl.info.

- Defining a modeling process that allows a seamless integration of, and interoperability between formalisms, and cover concerns of the cyber-physical system under consideration (mechanical, electrical, software, ...);
- Enforcing model interoperabililty, both at syntactic (e.g. refinement of a SysML block diagram as an AADL component, or integration of a SCADE node as an AADL entity); and semantics level, to guarantee that execution assumptions (model of time, propagation of events, data flow, properties) match across levels.

Should this challenge be solved, this would greatly ease communication across teams, each of which using its own reference tools, while allowing seamless models exchange.

Yet, the threat is that, as of today, both the modeling process and semantic interoperability remains open issues; syntactic interoperability being partly addressed in [9]. Hence, walls still exist between domains.

B. Heterogeneous models as outputs

The diversity in AADL model processing tools (formalisms, development approach, coverage of the language).

As part of the Ocarina project, we developed many AADL processing tools, targeting code generation, scheduling, model checking, WCET analysis. We took advantage of knowledge of code generation patterns enforced in our first Ada backend to enforce the same modeling patterns as inputs, and generate equivalent runtime systems.

This initiative remained isolated. Many other processing tools exist, hence raising the following challenges:

- Coverage of the modeling patterns: AADL is quite general and allows the expression of one pattern in multiple ways. Model processor usually perform a visitor pattern to map entities onto the destination formalism. Hence, one pattern may not be recognized by a particular tool. This could either trigger an error, or produce an output that does not reproduce all initial model elements;
- Compatibility of the outputs: how to compare the fidelity of results, and ensuring tat the correct tool is used for the correct analysis? For instance, authors in [39] illustrate how complex it is to select and implement the correct test for scheduling analysis.

Here, threat and opportunty are dual: more analysis imply more confidence in the system, yet inconsistent results – if detected – reduce confidence in the whole modeling process. A significant effort should be put on qualifying model processor so as to assert that they do not induce false or inconsistent results. Furthermore, the combination of tools should be evaluated: a set of guidelines shall be established to ensure consistencies in analysis.

C. Library of models

Rich and descriptive models are required to feed all analysis. AADL proposes a rich property mechanisms, with pre-defined properties and the capability to define new ones.

Yet, we note the following:

- there is no public repository of models reusable for typical blocks: e.g. a x86 processor, an ethernet network, FFT or PID building blocks, product lines, etc. Such description should be expressive enough to cover a wide range of analysis: e.g. behavior of a network device, power consumption of a CPU, etc.
- the coverage of default property sets is not complete, and different property sets have been defined separately to cover similar concerns.

These challenges are recurrent in the whole MBE community: we note a great emphasis has been put on defining modeling notations and tools. Yet, limited efforts have been defined to build a library of models, as we can see for instance in EDA tools.

Still, these blocks are required to perform high-fidelity analysis. Lack of model availability diverts engineers from MBE. Besides, one can note that many elements are not confidential, and could definitely be defined as part of a community-driven effort.

D. Training and Education

CPS and MBE tool support have the capability to shift engineers focus from implementation activities to system or functional definition of CPS. This is one of the key driver of the TASTE toolset in which we are involved with ESA and its partners: intensive code generation from heterogeneous functional models (SDL, SCADE, Simulink), integrated as AADL subprograms in an AADL architectural model provides a rapid prototyping platform for CPS. One may then focus on the system architecture and its non-functional properties, and how to validate/verify them.

Hence, engineers do not need to devote time to manual code integration. This raises some interesting questions regarding training and education of engineers.

At ISAE, our focus in on the education of engineers for the aeronautics and space domains. The challenge we face regularly is to upgrade our curriculum to match new design techniques, while preserving enough theoretical knowledge to master all steps that would be automated by a MDE toolchain for CPS. Yet, there is a temptation to reduce the amount of training and focus only on tools as a user, and trust the toolset to perform all analysis.

Discussion from the previous sections discourage us for taking such path: system must be mastered in all their dimensions. This requires actually a more intensive and combined training to know what to model, how to model and finally how to analyze it. Hence, engineering of Cyber-Physical Systems should focus more on system engineering, while Embedded Systems focus is on low-level design and implementation. Yet, it is unclear whether training in CPS is an extension to, or a different curriculum to Embedded System curricula.

E. Tailoring both modeling and analysis processes

MBE, and more specifically UML, taught us to model systems, and encourage us to define our own modeling process and to tailor it to the company business logic. We claim that modeling has a limited value if it is not backed by a verification process. AADL proposes a full set of analysis for supporting V&V. The Challenge here is many-fold:

- define boundaries between modeling and verification activities: when is a model complete enough for a particular validation? how to avoid overhead or contradiction in specifications?
- combine verification activities efficiently: the output of one tool may be the input of another, e.g. combining WCET and scheduling analysis; security and safety analysis. Dependencies between analysis must be traced so that model modifications will reduce verification activities to the analysis impacted by this change;
- qualify a dedicated process to match normative constraints; e.g. in the context of the MBE supplement to the DO-178C for developing avionics systems.

This challenge requires a particular understanding of analysis methods, the modeling notation (syntax, semantics, modeling pattern for analysis) and the system under consideration. A combined modeling and analysis process has the opportunity to reduce the complexity of designing a CPS. Existing tools around the AADL already allow for particular virtual integration of components and their analysis for one class of properties. This must be extended to multiple dimensions to cope with requirements for CPS.

This is perhaps the biggest threat to the applicability of model-based notations such as the AADL for the design of CPS: the current lack of integrated toolset, combining efficient model manipulation and V&V tool in a seamless process is a show stopper for many practitioners from the industry. The OPEES project ("Open Platform for the Engineering of Embedded Systems" – http://www.opees.org/) aims at defining not only such a platform, but also the ecosystem of companies that would make it a living community that enriches the set of tools required for CPS. Yet, this is to be confirmed by its partners.

VI. Conclusion

Cyber-Physical Systems requirements impose a radical shift in the way embedded systems are designed: one need to integrate several heterogeneous domains to support all design activities, from early requirement elicitation to detailed design, implementation and qualification activities.

In the mean time, several projects explored the applications of Model-Based Engineering to the design of CPS, covering most of these activities.

In this position paper, we advocated that the use of model-based engineering is not a solution, but rather an option to help designers. Taking AADL as a candidate to design CPS, we provided a comprehensive (yet partial and incomplete) list of projects that propose analysis of AADL models. We have shown that the scope of concerns covered by the AADL is quite large. Several projects demonstrated the feasibility of virtual integration of AADL building blocks for designing large systems.

We note that most of these initiatives follow a problem-centric view of the design of CPS, providing partial solutions to well-defined problems, e.g. to assess schedulability, dependability, performance, etc. Yet, CPS requires a tight interaction between domains.

We defined several challenges to be addressed by model-based and the AADL community to enable such interaction:

- support wider interoperability between heterogeneous model notations, at syntactic and semantics levels;
- define model patterns for combined analysis, allowing one single source model to be analyzed by several tools;
- publish a library of reusable models, to serve as basic building blocks for new designs;
- support training and education of CPS designer, to master inherent complexity of the modeling framework and associated tools;
- tailor of the modeling and verification processes to match project requirements.

These challenges are associated to many research opportunities and threats. We claim that these five challenges shall be addressed together to actually build a cyber-physical system around models, and transition from reseach to industry.

Acknowledgments

The author thanks the members of the AS2-C committee on the AADL, and members of Telecom ParisTech, LIP6/ MoVe, Université de Brest Occidentale, European Space Agency, Ellidiss, Rockwell Collins for their valuable support for AADL-related activities.

References

[1] Robert Allen, Steve Vestal, Dennis Cornhill, and Bruce Lewis. Using an architecture description language for quantitative analysis of real-time systems. In *Proceedings of the 3rd international workshop on Software and performance*, WOSP '02, pages 203–210, New York, NY, USA, 2002. ACM.

[2] Razieh Behjati, Tao Yue, Shiva Nejati, Lionel Briand, and Bran Selic. An aadl-based sysml profile for architecture level systems engineering: Approach, metamodels, and experiments. Technical Report 2011-03, Simula Research Laboratory, 2011.

[3] Bernard Berthomieu, Hubert Garavel, Frédéric Lang, and François Vernadat. Verifying dynamic properties of industrial critical systems using topcased/fiacre. *ERCIM News*, 2008(75), 2008.

[4] Jean-Paul Bodeveix, Raphaël Cavallero, David Chemouil, Mamoun Filali, and Jean-François Rolland. A mapping from aadl to java-rtsj. In Gregory Bollella, editor, *JTRES*, ACM International Conference Proceeding Series, pages 165–174. ACM, 2007.

[5] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. The compass approach: Correctness, modelling and performability of aerospace systems. In *Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security*, SAFECOMP '09, pages 173–186, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] Mohamed Yassin Chkouri, Anne Robert, Marius Bozga, and Joseph Sifakis. Translating aadl into bip - application to the verification of real-time systems. In Michel R. V. Chaudron, editor, *MoDELS Workshops*, volume 5421 of *Lecture Notes in Computer Science*, pages 5–19. Springer, 2008.

[7] Eric Conquet, Maxime Perrotin, Pierre Dissaux, Thanassis Tsiodras, and Jerome Hugues. The TASTE Toolset: turning human designed heterogeneous systems into computer built homogeneous software . In *Proceedings of Embedded Real Time Software and Systems 2010*, Toulouse, France, May 2010.

[8] Dionisio de Niz and Peter H. Feiler. On resource allocation in architectural models. In *ISORC*, pages 291–297. IEEE Computer Society, 2008.

[9] Julien Delange, Jerome Hugues, Laurent Pautet, and Diosisi o de Niz. A MDE-based Process for the Design, Implementation and Validation of Safety Critical Systems. In *Proceedings of the 5th UML& AADL Workshop (UML&AADL 2010)*, pages 319–324, University of Oxford, UK, March 2010.

[10] Julien Delange, Laurent Pautet, and Fabrice Kordon. Design, Verification and Implementation of MILS systems. In *Proceedings of the 21th International Symposium on Rapid System Prototyping*, pages 1–8, Fairfax, June 2010. IEEE Computer Society. MoVe INT LIP6.

[11] P. Dissaux. Using the AADL for mission critical software development. *2nd European Congress ERTS, EMBEDDED REAL TIME SOFTWARE Toulouse*, January 2004.

[12] P. Farail, P. Gaufillet, A. Canals, C. Le Camus, D. Sciamma, P. Michel, X. Crégut, and M. Pantel. TOPCASED : An Open Source Development Environment for Embedded Systems. *Chapter 11, From MDD Concepts to Experiments and Illustrations, ISTE Editor*, pages 195–207, September 2006.

[13] Madeleine Faugere, Thimothee Bourbeau, Robert de Simone, and Sebastien Gerard. Marte: Also an uml profile for modeling aadl applications. *Engineering of Complex Computer Systems, IEEE International Conference on*, 0:359–364, 2007.

[14] Peter Feiler, Joergen Hansson, and Dioniio de Niz. System Architecture Virtual Integration: An Industrial Case Study. Technical report, Software Engineering Institue, Carnegie Mellon University, 2009.

[15] Peter H. Feiler. Efficient embedded runtime systems through port communication optimization. In *ICECCS*, pages 294–300. IEEE Computer Society, 2008.

[16] Olivier Gilles and J<E9>r<F4>me Hugues. Towards Model-based optimisations of Real-Time systems, an application with the AADL. In *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2009)*, Pekin, Chine, August 2009.

[17] Olivier Gilles and Jerome Hugues. Applying WCET analysis at architectural level. In *Worst-Case Execution Time (WCET'08)*, pages 113–122, Prague, Czech Republic, July 2008.

[18] Olivier Gilles and Jerome Hugues. Expressing and enforcing user-defined constraints of AADL models. In *Proceedings of the 5th UML& AADL Workshop (UML&AADL 2010)*, pages 337–342, University of Oxford, UK, March 2010.

[19] Jorgen Hansson, Bruce Lewis, Jerome Hugues, Lutz Wrage, Peter Feiler, and John Morley. Model-Based Verification of Security and Non-Functional Behavior using AADL. *IEEE Security & Privacy*, 8(1):43–49, January 2010.

[20] Myron Hecht, Alexander Lam, and Chris Vogl. A tool set for integrated software and hardware dependability analysis using the architecture analysis and design language (aadl) and error model annex. In Perseil et al. [31], pages 361–366.

[21] Erwan Jahier, Nicolas Halbwachs, Pascal Raymond, Xavier Nicollin, and David Lesens. Virtual Execution of AADL Models via a Translation into Synchronous Programs. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 134 – 143, Salzburg, Autriche, 2007. ASSERT.

[22] Andreas Johnsen and Kristina Lundqvist. Developing dependable software-intensive systems: Aadl vs. east-adl. In A. Romanovsky and T. Vardanega, editors, *Ada-Europe 2011*, pages 103–117. Springer-Verlag, June 2011.

[23] Alexey V. Khoroshilov, Igor Koverninskiy, Alexandre Petrenko, and Alexander Ugnenko. Integrating aadl-based tool chain into existing industrial processes. In Perseil et al. [31], pages 367–371.

[24] Gilles Lasnier, Bechir Zalila, Laurent Pautet, and JÃ©rÃ´me Hugues. OCARINA: An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications. In *Reliable Software Technologies'09 - Ada Europe*, volume LNCS, pages 237–250, Brest, France, June 2009.

[25] Edward A. Lee. Cyber-physical systems - are computing foundations adequate? In *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, October 2006.

[26] Yue Ma, Huafeng Yu, Thierry Gautier, Jean-Pierre Talpin, Loïc Besnard, and Paul Le Guernic. System Synthesis from AADL using Polychrony. In *Electronic System Level Synthesis Conference*, San Diego, California, États-Unis, June 2011.

[27] Min-Young Nam, Rodolfo Pellizzoni, Lui Sha, and Richard M. Bradford. Asiist: Application specific i/o integration support tool for real-time bus architecture designs. In *ICECCS*, pages 11–22. IEEE Computer Society, 2009.

[28] Peter Csaba Ölveczky, Artur Boronat, and José Meseguer. Formal semantics and analysis of behavioral aadl models in real-time maude. In John Hatcliff and Elena Zucca, editors, *FMOODS/FORTE*, volume 6117 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2010.

[29] OMG. *OMG Systems Modeling Language (OMG SysML)*. OMG Document Number: formal/2010-06-01, June 2010.

[30] OMG. *A UML Profile for MARTE, version 1.1*. OMG Document Number: formal/2011-06-02, June 2011.

[31] Isabelle Perseil, Karin Breitman, and Roy Sterritt, editors. *16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2011, Las Vegas, Nevada, USA, 27-29 April 2011*. IEEE Computer Society, 2011.

[32] Lei Pi, Jean-Paul Bodeveix, and Mamoun Filali. Modeling aadl data communication with bip. In Fabrice Kordon and Yvon Kermarrec, editors, *Ada-Europe*, volume 5570 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2009.

[33] Ragunathan (Raj) Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 731–736, New York, NY, USA, 2010. ACM.

[34] Xavier Renault, Fabrice Kordon, and Jerome Hugues. Adapting models to model checkers, a case study : Analysing AADL using Time or Colored Petri Nets. In *IEEE/IFIP 20th International Sypmosium on Rapid System Prototyping* , Paris, France, June 2009.

[35] Ana-Elena Rugina, Karama Kanoun, and Mohamed Kaâniche. The adapt tool: From aadl architectural models to stochastic petri nets through model transformation. In *EDCC*, pages 85–90. IEEE Computer Society, 2008.

[36] SAE. Architecture Analysis and Design Language (AADL) AS-5506A. Technical report, The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 2.0, January 2009.

[37] SEI. OSATE : An extensible Source AADL Tool Environment. *SEI AADL Team technical Report*, December 2004.

[38] F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand. Investigating the usability of real-time scheduling theory with the Cheddar project. *Journal of Real-Time Systems, Springer Verlag*, 43(3):259–295, November 2009.

[39] Frank Singhoff, Alain Plantec, Pierre Dissaux, and Jérôme Legrand. Investigating the usability of real-time scheduling theory with the cheddar project. *Real-Time Systems*, 43(3):259–295, 2009.

[40] O. Sokolsky, I. Lee, and D. Clark. Schedulability Analysis of AADL models . International Parallel and Distributed Processing Symposium, IPDPS 2006,, April 2006.

[41] Roberto Varona-Gomez and Eugenio Villar. Aadl simulation and performance analysis in systemc. In *Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems*, ICECCS '09, pages 323–328, Washington, DC, USA, 2009. IEEE Computer Society.

[42] Zhibin Yang, Kai Hu, Jean-Paul Bodeveix, Lei Pi, Dianfu Ma, and Jean-Pierre Talpin. Two formal semantics of a subset of the aadl. In Perseil et al. [31], pages 344–349.

From Abstract Component Descriptions to Timed I/O-Interfaces in AUTOSAR

Stefan Neumann, Sebastian Wätzoldt and Holger Giese
Hasso Plattner Institute for Software Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3
Potsdam, Germany
Email: {stefan.neumann|sebastian.waetzoldt|holger.giese}@hpi.uni-potsdam.de

***Abstract*—When developing complex solutions with the AUTOSAR standard, the integration of components delivered from different suppliers is a crucial step. While the upfront agreement on syntactical interfaces can ensure syntactical conformance, other aspects such as timing or resource consumption have to be addressed rather late during integration and often require that the suppliers release IP relevant details. In this paper, we propose a first step to overcome this problem. We show how automated abstraction and checks can be applied to derive abstract timed interfaces for AUTOSAR ports from timed models of the component, which hide the IP relevant details while still providing enough information to exclude any invalid behavior when properly integrated.**

I. Introduction

Most innovations in the automotive domain are currently realized by software resulting in a dramatically increasing complexity of the developed systems. The **AUT**omotive **O**pen **S**ystem **AR**chitecture (AUTOSAR) framework has been established by car manufacturers (OEMs) as well as suppliers to deal with this complexity at the architectural level. In AUTOSAR, so-called software components (SWCs) are used to support a modular development approach for individual software parts. This modular approach supports the integration of software from different stakeholders without disclosing implementation details and, thus, allowing to protect intellectual property (IP).[1] AUTOSAR facilitates the parallel development of individual SWCs that can be integrated into the overall system later on based only on their interfaces (cf. [1]).

However, on the one hand, modularity helps to cope with the complexity and IP problem during development but, on the other hand, new problems during the component integration arise. While AUTOSAR supports well defined interfaces for syntactical aspects to allow such a modular development process, the standard only offers a timing profile for non-functional and timing aspects to specify required or guaranteed behavior [2, 3], but no concept for modular specification. Consequently, existing timing analysis approaches for an AUTOSAR compliant architecture like [4] do not fully preserve the modularity of SWCs. Internal details like execution times of functionality included in SWCs, execution orders, and dependencies concerning the data flow need to be available. Such details are often relevant for the implementation and, thus, object of IP. Furthermore, the analysis of timing properties has to be done on the overall system and not in a modular fashion.

In this work, we employ timed automata (TA) to investigate how to derive a timed interface description from the timed component model, which, on the one hand, supports abstraction for protecting IP and, on the other hand, permits to exclude that necessary local properties of the SWC like deadlock freedom or end-to-end timing constraints are compromised when integrated. For deriving an interface we apply abstraction like proposed in [5] on the timed model describing the SWC. Furthermore, a check is presented that reports a counterexample for a necessary local property in case the interface cannot guarantee them.

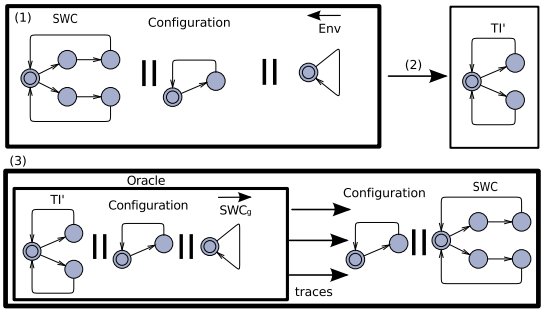


Figure 1: Steps of the approach and used models.

The approach consists of the following steps (see also Figure 1): (1) A TA for the timed behavior of the SWC is modeled that also considers configuration parameters related to the execution environment such as periods and execution orders of the functionality included in the SWC. (2) Then, we apply abstraction according to [5] to derive the desired abstract interface. The goal is to derive an interface that only provides interaction schemes that lead to a valid behavior of the SWC. (3) Because step 2 cannot guarantee by construction that all necessary information has been preserved, we outline how the derived interface can be checked against the original detailed TA model to decide whether the resulting interface only defines interactions leading to a valid state of the initial SWC. In case step 3 reports a counterexample, we have to rerun step 2 guided by the counterexample until

[1]In the automotive domain, OEMs as well as suppliers are unwilling to provide insights into internal details related to intellectual property.

a valid abstraction is found.

The paper is organized as follow: In Section II we briefly introduce the application example of a SWC for controlling a mobile robot. Subsequently, in Section III, we show how a TA model is derived in step 1, which describes the possible behavior of the previously introduced SWC. In Section IV, we apply the abstraction step 2 on the created TA model according to [5] and derive the desired abstract interface. In Section V, we investigate how we can check in step 3 whether the resulting interface only defines interactions leading to a valid valid behavior defined by the initial SWC based on the derived interface in combination with the original detailed TA model. The paper closes with a discussion of related work and a final conclusion.

II. Application Example: Mobile Robot

We built an AUTOSAR conform software architecture for the Robotino robot (http://www.robotino.com). The robot provides different types of sensor and actuator units. In the following example, only two of them are required to realize a feedback loop. The first type exists in the form of three drive units, realizing an omni-directional drive, which allows driving in any direction. Each drive unit consists of an actuator in the form of an electric motor as well as of a sensor in the form of an incremental encoder for measuring the actual rounds per minute of each motor. The second sensor type is provided in the form of distance sensors, which are able to measure the actual distance to obstacles.

A tool chain has been realized, including an existing real-time operating system (RTAI Linux, see https://www.rtai.org), which allows developing software according to the AUTOSAR standard. In this tool chain, we use MATLAB/Simulink (http://www.mathworks.com) to develop the control functions in combination with TargetLink for code generation as well as SystemDesk (http://www.dspace.com) for modeling, simulation, and generation of the AUTOSAR architecture. While conformance tests of syntactical aspects of interfaces are supported by the tool chain, this is not the case for timing aspects. The tool chain is used for an AUTOSAR conform development process and a more detailed description can be found in [6].

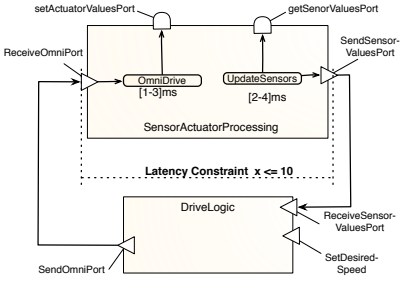


Figure 2: The two AUTOSAR SWCs for controlling the robot.

Like depicted in Figure 2, we use two SWCs *SensorActuatorProcessing* and *DriveLogic* in our application example, while only the first is described concerning its internal details. Such AUTOSAR SWCs communicate over ports with other SWCs, e.g., by reading or writing data values from or to such a port. The SWC at the top of Figure 2 exchanges signals via a sender-receiver port (triangle ports) as well applies function calls on service ports (rounded ones). Service ports provide functions from other SWCs. In AUTOSAR, a communication interface associated with a port describes, e.g., the type of messages, which can be sent or received using the port.

Furthermore, the internal behavior of the *SensorActuatorProcessing* SWC is modeled via the two Runnables *OmniDrive* and *UpdateSensors*. Runnables in AUTOSAR represent implementations, e.g., provided in the form of C-functions. The former Runnable reads the desired movement speed of the *DriveLogic* component and sets appropriate control commands to the actuators by using a service port. The latter reads the actual sensor data and sends them back to the *DriveLogic* SWC.

Additionally, Figure 2 shows some non-functional properties like an end-to-end latency constraint and the worst case (WCET) respectively best case execution times (BCET) of the Runnables.[2] The latency constraint requires that the time interval between an ingoing drive command and the point in time until actual sensor values are sent to the *SendSensorValuesPort* port, always needs to be smaller or equal to 10 ms. In addition, we claim that always both Runnables have to be executed in the same order and with a period of 10 ms.[3]

More information about the AUTOSAR standard and the semantics of the used component model can be found on the official website (http://www.autosar.org). In the next section, we derive a TA model representing the resulting behavior including timing aspects for the SWC *SensorActuatorProcessing*.

III. State-Based Behavior Models

In the following, we create a set of TA reflecting all information available in the current state of the development process from the perspective of the stakeholder developing the SWC *SensorActuatorProcessing*. Therefore, we can assume a white box view with available information about Runnables, execution orders, worst case, and best case execution times. In the present work, the TA model is created manually while for future work we are planning to investigate how an automatic derivation of the model can be

[2]Execution times are derived based on the C-functions associated with the Runnables. The required time for calling the functions provided by the service ports are included in the WCET/BCETs. For more accurate values also WCET analyzers can be potentially used.

[3]The required periodical execution is derived on the physical model (possible movement speed in combination with the distance sensor range) and is realized later on by the configuration in the form of modeled tasks.

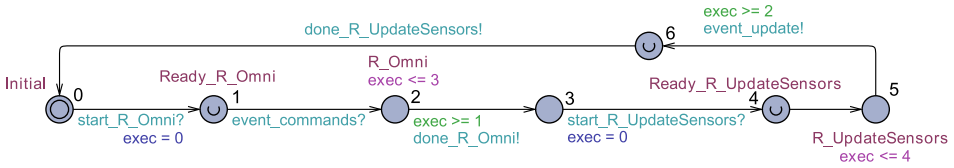


Figure 3: Automaton representing the abstract SWC.

achieved based on a given white box view. In Definition III.1 the formal semantics of the used timed automata model are given. For more information concerning TA cf. [7, 8].

Definition III.1 *A timed-automaton A is a 6-Tuple $A := (\Sigma, \mathcal{S}, \mathcal{S}^0, X, I, T)$, where Σ is the finite set of input and output signals including $\tau \in \Sigma$ representing an empty signal, $\mathcal{S}$ is a finite set of locations, $\mathcal{S}^0 \subseteq \mathcal{S}$ the initial location, $X := (x_1, .., x_n)$ a finite set of clock variables with $x_i \geqslant 0$, I a function $I \rightarrow \mathcal{C}(X)$, which assigns to each location a set of equations about clock variables called invariants and T is the set of transitions. $\mathcal{C}(X)$ is a set of conditions about the clock variables of X. $\mathcal{C}(X)$ consists of a set of equations of the form $x_i \prec c \vee c \prec x_i$, while $\prec$ rather is $<$ or $\leqslant$ and $c \in \mathbb{N}^+$. T is the set of transitions of the form: $T \subseteq \mathcal{S} \times \Sigma \times \mathcal{C}(X) \times 2^X \times \mathcal{S}$. A transition from location s to s' is defined via the tuple $(s, a, \varphi, \lambda, s')$, where $a \in \Sigma$ is the signal sent or received by the associated edge, which need to be sent or received by at least one other timed-automaton and φ is a set of constraints which need to be fulfilled as a precondition to trigger a transition. Each $\varphi_i \in \varphi$ is a constraint over clock variables of the form $x_i \prec c \vee c \prec x_i$, while $\prec$ rather is $<$ or $\leqslant$ with $c \in \mathbb{N}^+$. $\lambda \subseteq X$ is a set of clock variables that are reset to the value 0 in the case the transition is triggered.*

The *SensorActuatorProcessing* SWC at the top of Figure 2 provides two Runnables. The first one consumes an event that has been written from another SWC into a buffer. The second Runnable writes an event to a buffer, which can be read by another SWC. Each Runnable is transformed into one TA location.[4] Accordingly, the automaton has a *R_Omni* (RO) and *R_UpdateSensors* (RU) location. The resulting automaton including timing information is shown in Figure 3. For a better understanding, we have numbered the locations of the following TA, thus, each location has an assigned number allowing to reference locations based on it.

The BCET and WCET of each Runnable are reflected by an invariant defined for the corresponding location (e.g. *exec <= 4* in location 5) and a guard condition for leaving this location (e.g. *exec >= 2* at transition $5 \rightarrow 6$).

Each Runnable can be triggered by one or multiple OS tasks in an arbitrary order. This trigger is modeled as a *start* signal, which leads to a state change in the TA from a waiting to a ready location (cf. transition $0 \rightarrow 1$ and $3 \rightarrow 4$). After triggering, each Runnable starts immediately with its execution and notifies the task about completion via a *done* signal. The execution order of the Runnables can be different, depending on the order in which they are mapped to an OS task. One argument for including alternatives for the later implementation is to investigate whether execution orders have impact on timing properties. Therefore, alternating execution orders exist due to abstraction and are normally removed during development. The automaton in Figure 3 shows only one possible execution trace (Runnable RO is always executed before RU) to keep the model more compact and allow a better understanding.

AUTOSAR events are mapped to signals in the TA using synchronous channels. In the example, RO reads an event *commands* immediately after activation and processes the data afterwards. Furthermore, RU sends an *update* event after execution. Of course, both Runnables can read/send the event at every point in time during their execution. Therefore, the mapping is an approximation, which guarantees the availability of all needed events before, respectively after, the execution of a Runnable.

We can summarize the mapping of an AUTOSAR SWC to a TA as follows: each Runnable must be activated by a start signal from a task. It becomes ready and consumes/reads appropriate data from the input buffer (e.g. transition $1 \rightarrow 2$) and immediately starts execution. The last activity in the execution step is writing the output signal to a buffer (e.g. transition $5 \rightarrow 6$) and sending the *done* signal to the task.

The triggering of the Runnables inside an AUTOSAR SWC by OS tasks is crucial for the overall timing. Thus, tasks are also considered in our model at an abstract level. The tasks can be understood as a contract that needs to be considered when the SWC is integrated into a system later on. For our application example, the artifacts resulting from the Runnables (e.g., the resulting binaries when compiling and linking the C-Functions associated with the Runnables) need to be scheduled later on as defined by the tasks.

To preserve abstraction, the task description does not define exact start and end points. This allows the usage of the SWC in several different systems. Therefore, we create two TA representing two abstract task descriptions, which provide scheduling information. Each task (cf. Figure 4) consists of an initial (0) and ready (1) location. The task can leave the initial location if the CPU resource (modeled as a Boolean guard) is available and if the execution is permitted concerning the actual time slice of the overall period. Like

[4]One location for each Runnable with timing information is sufficient to apply a timing analysis. Each single location can be split into multiple, e.g., if the behavior of the Runnable is known in more detail.

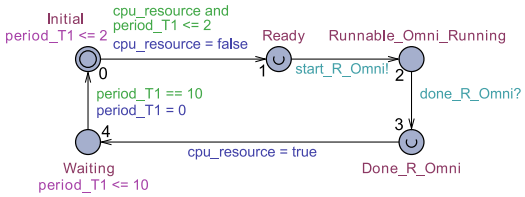


Figure 4: A corresponding task automaton of the OS configuration/contract.

shown in Figure 4, the time slice is defined to be within zero and two time units (ms) of each 10 ms period.

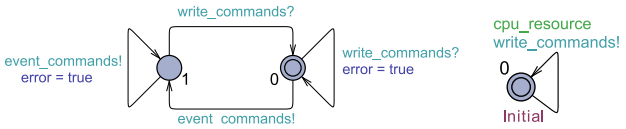
In the ready location, the task immediately (without consuming time, modeled in the form of an urgent location) activates the corresponding Runnable via a *start* signal (cf. transition $1 \rightarrow 2$). It waits until the Runnable indicates its completion by a *done* signal (see also Figure 3). If a task triggers more than one Runnable, the next Runnable is started immediately afterwards. In the end, the task releases the CPU resource and waits until the period is over.

We choose the configuration parameters as follows: two tasks are executed within a period of 10ms. The first task is executed in the first time slot of 5ms at the beginning of the period. Due to the WCET of the Runnable (cf. Fig. 3 loc. 2), it must be in the *Ready* location at least 2ms after the beginning of each period (10ms). The second task is executed within the second half of the period and must be activated at least 4ms before the period is over. The AUTOSAR OS provides mechanisms for the triggering of OS tasks based on a period as well as a first absolute activation time via so-called OS alarms. Thus, the described task configuration can be implemented in an AUTOSAR conform system later on.

AUTOSAR allows different forms of exchanging data with synchronous or asynchronous communication. In this work, we focus on asynchronous communication via buffered events. Accordingly, for each event that is provided or required by a port, we need to model the buffer explicitly. The TA in Figure 5(a) represents the buffer for the *command* event, which is received by the *ReceiveOmniPort* (cf. Fig. 2) and processed by the Runnable RO. Each buffer has a size of one.[5] The Runnable can process exactly one *command* signal after receiving it from the environment, e.g., from another SWC. If such a signal is sent or processed twice, an error variable is set to true indicating an invalid state of the SWC. To be able to explore the state space of the created TA model, we also need a representation of the environment, or in other words someone needs to send respectively consume signals from/to the SWC (the buffer). Because we want restrict the possible behavior for the SWC in any way, we create the most generic environment that can exist. Such

[5]It is also possible to model buffers of different size but the used application example of the robot requires buffers of size one.

an environment is able to send/receive a signal from/to the SWC in each and every state. The only restriction we made is that such a signal can be sent on a single-core CPU, like in the case of the robot, only if the corresponding resource (CPU) is available. The resulting TA representing such a generic environment is shown in Figure 5(b).



(a) Buffer for signal A.

(b) Input Automata.

Figure 5: Automata representing the input buffer and the generic environment for the *commands* event.

IV. Timed-Interface Automata

Now we are interested in deriving a behavior description that defines those interactions valid with the SWC while hiding internal details, e.g., the number of existing Runnables, dependencies concerning the data flow or estimated execution times of individual Runnables. We need to remove those details that are object of IP or that hinder a modular development process. For this purpose, we proceed as follows: (1) we create the full state space of the SWC, (2) we transfer the derived state space to an initial (not abstract) interface, and (3) we apply abstraction and derive an interface that reflects those traces of the SWC, which can be observed on the externally visible elements only.

In the first step, the basic idea is to explore the overall state space. For this purpose, we give a definition of a labeled transition system (LTS) that represents traces of one or multiple TA. This definition includes the formal construct of so-called clockzones, which are able to represent the symbolic representation of valid evaluations of clocks when one or multiple TA are in a specific global state.

Definition IV.1 *A Clockzone Z consists of a set X of clock variables x_i. Each x_i can take values of $\mathbb{R}^+ \cup 0$, while $i \in \mathbb{N}^+$ and $i > 0$. Additionally, a constant exists in the form of the clock x_0 with the value 0 as well as a set of constraints $c \in C$ in the form of equations $x_j \prec d, d \prec x_j$ or $x_i - x_j \prec d$, with $i, j \in \mathbb{N}^+$, $d \in \mathbb{Z}$ and $\prec \in \{<, \leqslant\}$. The clockzone is the result of the conjunction of all constraints that are given by C.*

Definition IV.2 *A labeled transition system representing sequences over the parallel construct of multiple timed-automata $A_0 \parallel \ldots \parallel A_n$ consists of the tuple $L^p := (\Sigma^p, Q^p, Q_0^p, X^p, T^p)$ where Σ^p is the set of input and output signals including all signals of $A_0 \ldots A_n$, X^p is the set of clock variables included in $A_0 \ldots A_n$, and T^p is the set of transitions of the form $T^p \subseteq \mathcal{Q}^p \times \Sigma^p \times \mathcal{Q}^p$. Each $q \in Q^p$ is a tuple $q := (s^p, z^p)$ representing the states of all parallel*

automata $A_0 \parallel ... \parallel A_n$ with $s^p \subset Q_0 \times ... \times Q_n$ where Q_i with i from 0 to n represents the locations of the automata A_i. z^p is the clockzone for a single state consisting of the union over all clock variables $X_0 \cup ... \cup X_i$ with X_i from automaton A_i and the possible clock assignments in this state. A transition from state q to q' with $q, q' \in Q^p$ is defined via the tuple $(q, a, \varphi, \lambda, q')$, where $a \in \Sigma^p$ is a signal, which needs to be sent or received by the timed-automata, φ is a set of constraints, and $\varphi_i \in \varphi$ is a constraint over clock variables of the form $x_i \prec c \vee c \prec x_i$, while $\prec$ rather is $<$ or $\leqslant$, $x_i \in X^p$ and $c \in \mathbb{N}^+$. $\lambda \subseteq X^p$ is a set of clocks that are reseted during the transition.

Due to the number of states resulting from a full state space exploration of our example application, to allow a better understanding as well as a more compact representation, we derive only one symbolic trace. It is described in the form of a LTS. For this purpose, we use the simulation capabilities provided by the UPPAAL tool. It allows us to explore the state space step-by-step and to manually choose between possible alternatives during exploration.[6] The result of such a simulation in UPPAAL is a symbolic trace representing a subset of the overall state space. We start the simulation with all involved TA in their initial location. At the beginning, the automaton representing task 1 (see Figure 4) switches within two time units to location *Ready* and the resource of the CPU is occupied. Because *Ready* is an urgent location and, thus, time is not allowed to pass. Task 1 switches immediately to its only successor location and triggers the execution of the associated Runnable represented by location *R_Omni* of the SWC (cf. Figure 3) via the *start_ R_Omni* signal. As a result, the value of the input buffer is read and the TA shown in Figure 5(a) switches to the location on the left, representing an empty buffer, respectively a buffer allowed to be overwritten. The SWC as well as task 1 switch simultaneously to their individual successor locations synchronized via the signal *done_R_Omni*. Because task 1 is again in an urgent location and no other transition in the TA model is enabled, it switches to location *Waiting* and the resource of the CPU becomes available. From this global state, we write a new value on the buffer and the TA of Figure 5(a) switches back to its initial location. During the above described simulation an error state has not been reached.

The observable behavior during the above described simulation in UPPAAL is a subset of the full state space and, thus, can also be represented in the form of an LTS. Due to space limitations, we do not show the parallel construct of all involved TA models shown in the Figures 3, 4 and 5. Instead, we encode each location of the involved TA as follows: Each individual TA has a set of locations, each numbered like shown in Figure 5(a) where the location on the left has index 1 and the location on the right has index 0. We represent the overall state of the parallel construct by encoding each location in the form of a value vector. The dimension of the vector is equal to the number of TA included in the parallel composition. Each position in the vector represents the current location of exactly one TA, e.g., the current location of the TA of the SWC shown in Figure 3 is reflected by the value at the first position of the vector, the location of the TA representing task 1 is stored on the second position, and the location of the input buffer on the third position. Due to the fact that all remaining TA stay in the same location (initial location) during the whole simulation, only a vector of dimension three is shown in Figure 6.

The first location of the automaton shown on the left of Figure 6 depicts the starting point of the previously described simulation where all involved TA stay in their initial location (index 0). The shown automaton represents a cutout of the LTS L^p representing the overall state space of the SWC. Thus, we are able to encode and visualize the discrete part of the global state L^P in a much more compact form. The discrete part s^p of a state $q \in Q^p$ with $q := (s^p, z^p)$ of the LTS L^p is described by such a vector and the continuous part z^p is defined by the possible clock valuations observable during simulation. The valid clock valuations are directly taken from the UPPAAL simulator representing the continuous part z^p. Guards, clock resets, urgent locations as well as sent signals are simply taken from the transitions of the TA model observed during simulation. Thus, the first state q_0 of the resulting LTS derived by the simulation consists of the discrete part ${s^p}_0 = 0, 0, 0$, the continuous part ${z^p}_0 = z_0$, with z_0 being the clockzone including constraints: $period\_T1 \geqslant 0 \wedge period\_T1 \leqslant 2 \wedge exec \geqslant 0 \wedge exec \leqslant 2 \wedge period\_T1 \leqslant exec \wedge period\_T \geqslant exec$. Sent or received signals, clock resets, and guards are taken from the simulation results accordingly.

After deriving the (partial) state space, we proceed with step two (2) and transform the created LTS to an initial interface. We start with an empty interface in the form of the TA $TI^{'} := (\Sigma^{'}, \mathcal{S}^{'}, \mathcal{S}^{0^{'}}, X^{'}, I^{'}, T^{'})$. For each state $q_i \in Q^p$ of the derived LTS with $q_i := ({s^p}_i, {s^z}_i)$, we define a copy s_i of the discrete part ${s^p}_i$ with the staying condition (as well as all outgoing transition guards) restricted to their intersection with ${s^z}_i$. We add each s_i to the set $\mathcal{S}^{'}$, the resulting staying condition to the function $I^{'}$ and the transition including the derived guards to $T^{'}$ accordingly. Transitions and states that become empty during this procedure are removed. For the simulated trace shown in Figure 6, we only need to transform the shown constraints on the possible clock valuations to appropriate invariants for the resulting locations.[7]

[6] For more information about UPPAAL and the provided simulator see [9].

[7] Several internal signals sent between constituent parts of the SWC observable during simulation are not shown in Figure 6 to allow a better understanding. In any case, these signals will be removed, at latest during the following described abstraction mechanism.

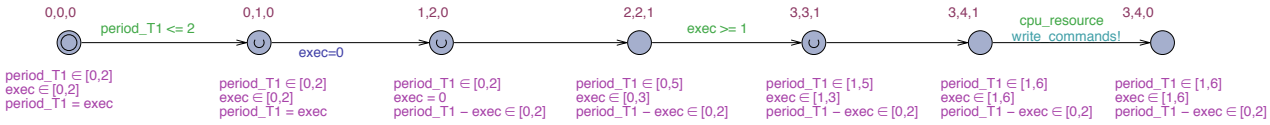


Figure 6: The symbolic trace derived by the UPPAAL simulator.

After deriving the initial interface, we need to hide those information not observable on the interface. We proceed with the third step (3) by applying abstraction according to [5]. In [5] the authors describe how an abstract representation of the timed input/output behavior is derived by hiding all internal clocks, variables, signals and so on, based on the behavior from the perspective of a white box view.

As a result, for our example only that information which is associated with the configuration or which can be observed from outside of the component is allowed to be preserved. This is the case for the clock that measures the defined period (assuming the period to be known). The following applied abstraction mechanism is sketched in listing 1: The procedure $deriveAbstractInterface$ takes the input $TI^{'}$ representing the full and not abstract interface, the set OS representing the observable signals, and OC representing the observable clocks. At first, the procedure iterates over all transitions of the interface (line 7) and removes all signals from the transitions that are not element of OS (line 9). Second, the procedure projects all constraints included in the guards of each transition on the remaining clocks, which are element of OC (line 12-16). Further, it removes all clock resets of clocks not in OC (line 18-22). In the next step, the procedure iterates over all states respectively locations (line 25). It takes the set of invariants assigned to each location via the function $I^{'}$ and projects the included constraints (invariants) on the remaining clocks of OC (see line 26). The last step (line 29) assigns only the observable clocks to the TA.

The projection used in line 14 and 26 works as follows: Assume we have two constraints of the form $1 \leqslant c_1 \leqslant 2 \wedge 1 \leqslant c_1 - c_2 \leqslant 2$ with the two clocks $c_1 \in OC$ and $c_2 \notin OC$. Removing clock c_2 and projecting the constraints on c_1 results in the constraint $2 \leqslant c_1 \leqslant 4$. For more details concerning the abstraction as well as the projection compare [5].

Now we apply the above sketched abstraction mechanism on the example taken from UPPAAL. In location $0,0,0$ of the automaton shown in Figure 6 all clocks have the same value but only clock *period_T* is externally visible. Thus, the clock *exec* can be removed and existing constraints are projected on the remaining clocks. The same holds for the location $0,1,0$.

Listing 1: Derivation of abstract interface

```
1  procedure deriveAbstractInterface()
2  parameter: TI' //Initial interface
3  ,OS//Observable signals
4  ,OC//Observable clocks
5
6  //Remove internal signals
7  foreach t ∈ T' with t := (q,a,φ,λ,q') do//For each transition
8    if a ∉ OS do//Signal not observable
9          removeSignal a
10   endif
11   //Project guards
12   foreach φ_i ∈ φ do//For each guard
13      if 'guard includes not observable clock' do
14         projectGuards
15      endif
16   endforeach
17   //Remove clock resets
18   foreach λ_i ∈ λ do//For each clock reset
19      if 'reset includes not observable clock' do
20         removeReset
21      endif
22   endforeach
23 endforeach
24 //Project invariants
25 foreach s ∈ S' do
26   project invariants of I'(s) on remaining clocks
27 endforeach
28 //Assign observable clocks
29 X' = OC
```

Additionally, we can collapse the resulting TA, e.g., the possible clock valuations in location $0,0,0$ in conjunction with the guard of the outgoing transition allow exactly the same clock valuations as possible in the only successor location $0,1,0$. Thus, we are able to collapse locations resulting in the automaton shown in Figure 7. For a complete description of the above sketched derivation of a interface, the applied abstraction mechanism as well as the reduction see [5].

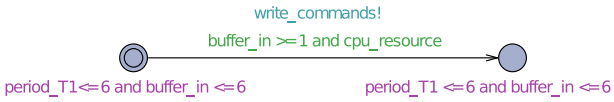


Figure 7: Reduced abstract trace.

V. Conformance between Interface and SWC

It can be seen that the derived interface of Figure 7 on its own is not closed anymore, due to the fact that the emitted signal *write_commands* needs to be consumed to take the transition. This is the case because the buffer, which consumes the signal, is part of the SWC and has been removed during abstraction. Again, we have a similar problem like already described in Section III where we

have created the generic environment. There, we needed a representation for the environment that is able to produce signals that can be consumed by the SWC. Now we have derived an interface $TI^{'}$ that defines, which signals can be sent from the environment to the SWC. Accordingly, we need a component consuming the signal. Thus, we create an abstract representation of the black box SWC SWC_g (see Figure 8), which is able to consume the signals provided by the interface $TI^{'}$ in each and every state.

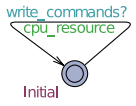


Figure 8: The abstract representation of the black box SWC SWC_g.

For being able to reason whether the composition with other SWCs leads to a valid behavior,[8] $TI^{'}$ shall be such that it can be used as an oracle. It provides only traces, which are valid in combination with the black box SWC in any case. In other words, the derived interface automaton in parallel with the most generic SWC is not allowed to provide interaction schemes, which potentially lead to a violation of the previously validated properties, like deadlock freedom, absence of error states, and required response times. For this purpose, we create the oracle in the form of the parallel product like sketched at the bottom of Figure 1 consisting of $TI^{'} \parallel Configuration \parallel SWC_g$. The configuration is represented in the form of the task shown in Figure 4.

We call *ORA* the set of all possible traces observable on the oracle. Assuming that we are able to divide all possible behavior of the original SWC (considered as a white box) into two different sets, the first VAL representing valid behavior and the second $INVAL$ representing invalid behavior, we can define more formally what is not allowed in relation between the traces of the SWC and those of the oracle:

$$\neg\exists t_i \in ORA, t_j \in INVAL : t_i = t_j \restriction \alpha(ORA)$$

In other words, if we are able to find a trace in the oracle that leads to an invalid state of the real SWC, the traces defined by the oracle (the interface) are not safe.

The restriction operator $\restriction$ on an alphabet $\alpha(a)$ is defined as follows: Applied on trace t it removes all actions, like signals, that are not in the given alphabet of a, where a can be a TA. Thus, we take trace t, consisting of multiple actions $(e_0, d_0)...(e_n, d_n)$, where e_i represents an signal and d_i the point in time on which the signal has been sent, and remove all actions from t that are not included in the alphabet of a.

We use the UPPAAL model checker again for a slightly modified model:

We create the parallel construct of the interface $TI^{'}$, with the original SWC, the configuration as well as a slightly modified form of the generic SWC. We add an additional location to the generic SWC, which is a committed location.[9] Thus, the generic SWC in parallel with the interface $TI^{'}$ creates the same set of traces compared to the generic SWC shown in Figure 8. Now we investigate whether all signals consumed by the generic (abstract) SWC can be consumed in the same way by the original (white box) SWC. For this purpose, we send a signal *write_commands_2* to the SWC each time the modified generic SWC has decided to receive signal *write_commands* from $TI^{'}$. Accordingly, the SWC is forced to receive the signal at the same point in time when the oracle is able to produce an signal, or in other words all signals generated by the oracle are instantaneously forwarded to the (white box) SWC.

Now, we can apply the same checks in the UPPAAL verifier, which we previously applied on the SWC in combination with the generic environment. If we are able to find an example where at least one property is violated, we have found a witness for the previously formally defined case and, thus, the desired modularity is not guaranteed when using the interface. If we are not able to find such a violation and all traces derived on the oracle are still valid for the SWC, we know that a trace like defined above does not exist and the SWC can be used in a fully modular fashion as a black box using $TI^{'}$.

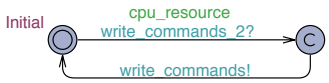


Figure 9: The modified generic SWC.

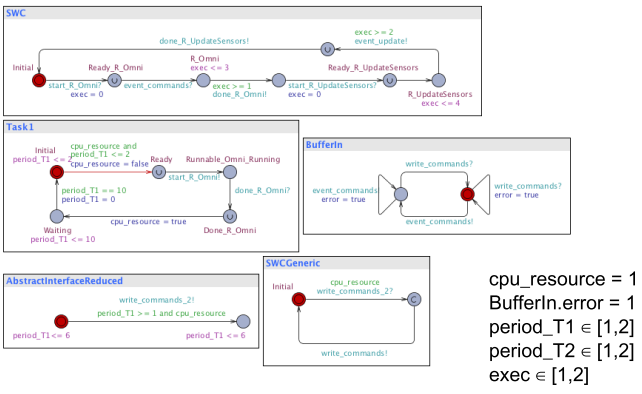


Figure 10: Reached counterexample in UPPAAL.

For our application example, we apply the described check for the derived automaton shown in Figure 7 representing a fragment of the state space of the interface $TI^{'}$. We ask the UPPAAL model checker whether a deadlock or an error state is reachable within six time units (the time period for which the fragment describes the derived valid interactions taken from the UPPAAL simulator). As a result UPPAAL creates a witness in the form of a trace where the input

[8] In our application example, we have derived the LTS based on valid traces only.

[9] A committed location in UPPAAL need to be left in the next step.

buffer is overwritten in the time interval between 1 and 2 time units. The associated state as well as the possible clock valuations are shown in Figure 10. Because the variable *BufferIn.error* has the value 1, representing an error state, we have found a witness indicating that we are not able to use the SWC as black box based on the given interface only. This is the case because the considered fragment of the interface already contains a trace leading to an error state. Within our application example we have been able to remove interaction schemes from the interface based on the shown counter examples derived by the UPPAAL model checker. Several different possibilities exist for removing such invalid interactions schemes from the interface, e.g., changing configuration parameters of the OS tasks or simply restricting the interface by removing traces. Nevertheless, each time changes are applied, e.g., on the configuration, we need to derive the new interface description and, thus, also need to reapply the activities described in Sections IV and V.

VI. Conclusion and Related Work

We have shown the technical feasibility of deriving a TA model from a white box view of an AUTOSAR SWC. We have further investigated how to derive an abstract interface description based on this TA model according to [5]. We have demonstrated that, on the one hand, some obstacles and pitfalls must be kept in mind when deriving such an interface, e.g., in the case of the found error states, but, on the other hand, we can automatically identify these pitfalls in the form of examples. At least for the considered SWC, we have been able to remove such pitfalls by modifying the configuration or removing traces from the interface. As a result we obtain a SWC that can be used in the desired modular fashion. Compared to existing approaches dealing with TA for describing the interface behavior of components, we believe that for the considered application domain such an automatic check between the white box SWC and the derived interface is necessary, due to the fact that assumptions and required preconditions of existing approaches are not fulfilled when deriving a TA based on a given AUTOSAR SWC. For example, in [10] timed-interfaces are defined to be compatible if there is at least the possibility that they can work together and, thus, undesired behavior is not necessarily excluded. In [11] deterministic TA and in [12] time deterministic TA are required for being able to reason, e.g., about time trace inclusion, and in [13] TA are required to be receptive, which is hard to achieve at the implementation level, especially for AUTOSAR SWCs. Due to the fact, that applying abstraction on the more detailed model is the main tool for protecting IPs, theories dealing with timed interfaces not supporting such an abstraction mechanism (e.g., [10, 13]) are not considered here.

For the future, we plan to evaluate in more detail, which aspect of an AUTOSAR SWC can be represented by a TA model based on a more complex application example. Furthermore, we are planing to realize the automatic derivation of the TA models as well as the automatic derivation of interfaces based on a given white box description of an AUTOSAR SWC. We will investigate whether and how it will be possible to automatically change the configuration part or to restrict the interfaces for the purpose of automatically achieving modularity like described in this work.

References

[1] H. Giese, S. Neumann, O. Niggemann, and B. Schätz, "Model-based integration," ser. MBEERTS'07. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 17–54.

[2] *AUTOSAR - Specification of Timing Extensions*, AUTOSAR GbR, 2010, version 1.1.0. [Online]. Available: http://www.autosar.org

[3] K. Richter, "The autosar timing model - status and challenges," in *Proceedings of the Second ISoLA*. Washington, DC, USA: IEEE Computer Society, 2006.

[4] K. Richter, N. Feiertag, M. Rudorfer, O. Scheickl, and C. Ainhauser, "How Timing Interfaces in AUTOSAR can Improve Distributed Development of Real-Time Software," in *GI Jahrestagung (2)*, 2008.

[5] R. B. Salah, M. Bozga, and O. Maler, "On timed components and their abstraction," in *Proceedings of SAVCBS '07*. ACM, 2007.

[6] B. Becker, S. Neumann, M. Schenk, A. Treffer, and H. Giese, "Model-Based Extension of AUTOSAR for Architectural Online Reconfiguration," in *Models in Software Engineering, Workshops and Symposia at MODELS 2009*, ser. LNCS. Springer, 2010.

[7] R. Alur and D. L. Dill, "A Theory of Timed Automata," in *Theoretical Computer Science*, vol. 126, no. 2, 1994.

[8] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools." Springer, 2004, pp. 87–124.

[9] G. Behrmann, A. David, and K. Larsen, "A Tutorial on Uppaal," in *SFM-04:RT*, ser. LNCS. Springer, 2004.

[10] L. Alfaro, T. A. Henzinger, and M. Stoelinga, "Timed Interfaces," in *EMSOFT '02*. London, UK: Springer, 2002, pp. 108–122.

[11] T. Bourke and A. Sowmya, "Automatically transforming and relating Uppaal models of embedded systems," in *EMSOFT '08*. NY, USA: ACM, 2008.

[12] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Methodologies for specification of real-time systems using timed I/O automata," in *Proceedings of the 8th international conference on Formal methods for components and objects*, ser. FMCO'09. Springer, 2010.

[13] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, *The Theory of Timed I/O Automata (Synthesis Lectures in Computer Science)*. Morgan & Claypool Publishers, 2006.

Quantitative Fault Propagation Analysis for Networked Cyber-Physical Systems

Linda Briesemeister,* Grit Denker,* Daniel Elenius,* Ian Mason,*
Srivatsan Varadarajan,† Devesh Bhatt,† Brendan Hall,† Gabor Madl,† and Wilfried Steiner‡
*SRI International, Menlo Park, CA, USA
†Honeywell Labs, Golden Valley, MN, USA
‡TTTech Computertechnik AG, Vienna, Austria
firstname.lastname@[sri|honeywell|tttech].com

***Abstract*—This paper presents an approach to analyzing a model of networked cyber-physical systems for fault propagation. We present an implementation of a probabilistic logic model, which allows for reasoning via symbolic evaluation as well as numeric evaluation to perform a quantitative fault analysis. Our models are built from a few building blocks, which can be instantiated as standard or high integrity; communication paths can be made redundant, and finally, whole subsystem blocks can be replicated. We assume an underlying networking infrastructure of TTEthernet, which allows traffic of time-triggered, rate-constrained, or best-effort modes with different safety features. We apply our approach to a case study of a brake-by-wire system that contains communication flows with different traffic modes according to their criticality.**

I. Introduction

Safety-critical networked systems such as the avionics in an airplane or an automotive X-by-wire system typically have to be fault tolerant, i.e., these systems have to remain operational even in the presence of failures. Achieving fault tolerance requires safety analysis that is best performed beginning with the earliest design stages. Part of such design analysis is the so-called fault propagation analysis of the system model. In safety-critical systems, we are interested in characterization faults and how faults manifest as failures that affect overall reliability goals of the system.

In this paper, we present a fault analysis tool for networked cyber-physical systems (CPSs). We present networked CPSs as a graph where nodes represent physical components, such as CPU, communication controller, and physical links, and edges represent the direct information flow between nodes. We assume that faults occurring in a particular node may propagate along the connections in the graph representing the networked cyber-physical systems. For example, a faulty communication controller may send an arbitrary number of faulty messages. Without having protection mechanisms in place, the failure of the communication controller could lead to monopolizing the shared network infrastructure, thereby making it unusable for other non-faulty communication controllers.

Various redundancy schemes and safety techniques can be used as a protection mechanism to prevent or mitigate failure propagation and, thus, improve system availability and integrity. However, the decision about where to locate what protection mechanism along the graph of the networked CPS currently relies heavily on expert knowledge. The proposed tool allows designers to efficiently analyze various network choices and evaluate the effect of protection mechanisms on fault propagation, and therefore rely less on expert knowledge.

We provide a formal Maude [4] model that uses basic building blocks (Host, End System, Switch) to model networked CPS topologies. Given an underlying networking infrastructure of TTEthernet (TTE) [15], which is an extension to standard Ethernet, we can select traffic modes with varying safety properties. Other safety techniques are selecting components with standard- or high-integrity features and adding path and system redundancy. We then combine this network and protection mechanism model with fault propagation rules that describe in a probabilistic logic how faults occurring in a component, or at an input of a component, transform into a fault at the output. Our implementation allows for both symbolic and quantitative fault propagation analysis. We use the example of a brake-by-wire (BBW) system with multiple communication flows to illustrate our approach.

II. Related Work

Our work is mostly inspired by the functional and component failure analysis of the Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [14] method. HiP-HOPS uses these analyses in conjunction with the system model to automatically construct fault trees. Our symbolic analysis yields probabilistic logic formulas, which could be interpreted as

fault trees. In contrast to our work, HiP-HOPS supports mature tool integration and hierarchical models.

An inductive method to study fault propagation is the Failure Propagation and Transformation Analysis (FPTA) [6]. A failure logic for each component allows for automated analysis of the system. Here, the failure logic depends only on the inputs, which is appropriate for modeling software failures, but cannot explicitly use failure states of the component itself. In [5], the authors present an approach to quantify the failure propagation analysis using the PRISM probabilistic model checker.

So-called State/Event-Fault Trees (SEFTs) [8] integrate finite state models with fault trees, in particular Markov Chains and Statecharts. For quantitative probabilistic analysis, SEFTs are translated component-wise into Dynamic Stochastic Petri Nets (DSPNs) and then merged into one flat net for analysis using tools such as TimeNet.

III. Networked System Architecture

For this study, we assume an underlying communication network using TTEthernet (TTE) technology [16]. TTE is an extension to standard Ethernet for usage in networked cyber-physical systems such as avionics or automotive applications. In standard Ethernet, messages are communicated according to a best-effort paradigm, which means that no bounds on a message's transmission latency can be given. Under high load, buffers in network switches can overflow and messages are lost entirely. In noncritical systems, higher-layer protocols such as TCP/IP often compensate for message loss using re-transmission strategies. However, networked cyber-physical systems typically have stringent end-to-end timing requirements that do not allow the temporal penalty of repeated transmission attempts. Furthermore, standard networks cannot guarantee that any successive transmissions will actually be successful.

TTE achieves timely transmissions with known upper bounds on latency and jitter by providing rate-constrained (RC) and time-triggered (TT) communication paradigms in addition to standard best-effort (BE) traffic. Rate-constrained traffic is well-known from an avionics Ethernet variant ARINC 664P7-1 [1]. It is based on an *a priori* agreement of the network components on the number, size, and maximum frequency of messages. This knowledge is sufficient to calculate the required network resources, i.e., the switch buffers, and it can be guaranteed that no message will be lost due to buffer overflows. However, different network components may source their messages at about the same point in time or may even source several messages back to back. This uncoordinated transmission pattern quickly leads to high transmission latencies.

The time-triggered communication paradigm takes the level of determinism to the extreme. Here, all communication participants are equipped with local clocks that are brought in close agreement to establish a synchronized global time. An Ethernet frame may now be dispatched at an *a priori* specified point in the global time, which makes the frame transfer time-triggered. The sum of all those specified points in time for dispatch, relay, and potentially also reception is collectively referred to as the "communication schedule" for time-triggered communication. When correctly aligned, the communication schedule ensures that any two time-triggered Ethernet frames will never compete for transmission resources (e.g., communication links, switch buffers) and their transmission latencies can be kept minimal and almost constant.

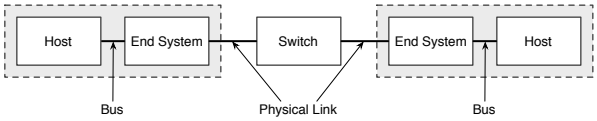


Fig. 1. Simple chain of network components

To model networked CPSs, we identified a few generic components from which we build networks. Fig. 1 shows an example of these components that are connected via a communication link. We distinguish the host (H) component (CPU, memory, I/O and so on) from an end system (ES), which denotes the network interface card, here the TTE controller. H and ES components are often located on the same physical platform and are usually connected via an on-chip bus. To connect different ESs in the network, one uses at least one switch (SW) in between. For our models, these are TTE SWs, which allow the different traffic modes, but a legacy system could also be built with standard Ethernet SWs. A bus connects Hs and ESs, and a physical link connects SWs with other SWs or ESs.

A. Dependability Means

Aside from choosing an appropriate traffic mode that TTE supports, we consider the following three broad fault tolerance strategies that improve the *integrity* and *availability* properties of a network architecture and thereby also enhance the safety of the network. Integrity is the absence of improper system alteration; it is the ability of a system to detect faults in its own operation and to reach a fail-safe state or safe output states in the event of failure and inability to recover. Availability is a measure that the system functions correctly in the presence or absence of both transient and permanent failures of the different network components. We discuss the three strategies next.

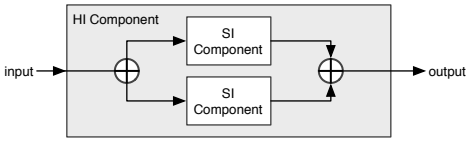


Fig. 2. High-integrity self-checking pair

In our model, all Hs, ESs, and SWs can be instantiated as **standard-** or **high-integrity** components. As shown in Fig. 2, a High-Integrity (HI) component consists of two replicated Standard-Integrity (SI) components functioning as a single unit for increased integrity. The input messages received at each SI component pair are exchanged and compared to ensure that each of them receive identical inputs and as a result triggers identical internal message processing states in each component. The output messages are also compared and only identical messages whereby both components agree are in effect transmitted. The expectation is that by providing each SI component with identical inputs, using the same internal processing logic, states and clocks, the pair's outputs will match exactly under fault-free conditions. When operators $\bigoplus$ during input exchange or output comparison do not agree, then the combined high-integrity component fails silently so as to not influence downstream components. In our formal model, this behavior translates into converting an arbitrary fault into an omission failure. Notice that this mechanism uses replication of components solely for integrity at the cost of availability.

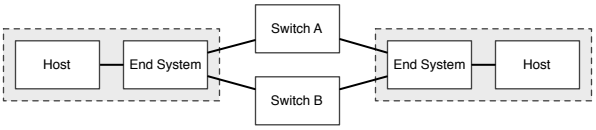


Fig. 3. Path redundancy in simple chain

Including **path redundancy** increases the availability of a message by transmitting an identical copy over disjoint but redundant paths from the source to the destination. Fig. 3 shows the simple chain of network components enhanced with two switches A and B that allow for disjoint paths between the ESs. The receiving ES now picks the first arriving message from the sender, thereby protecting against message loss on one of the paths due to permanent or transient failures of the SWs or links on that path. Note that this mechanism does not protect against failures at source or destination components.

Finally, **system redundancy** in conjunction with the availability and integrity constructs introduced above, can be used to improve the overall safety and reliability of the network, if safety properties like replicate group determinism or interactive consistency are strictly adhered to [2], [7], [9], [10]. Literature defines architectures such as dual-dual active standby, triple or N-modular redundancy, and triplex-triplex [3], [17]. Additional replication coordination mechanisms are passive, semi-active, or active replication; voting and multistage N or Triple Modular Redundancy (N(T)MR), congruency exchange.

B. Faults

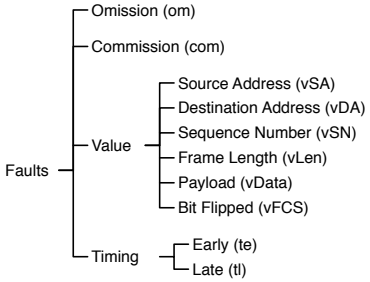


Fig. 4. Hierarchy of faults

We depict a hierarchy of faults in Fig. 4. This hierarchy is meant as a reference to typical faults used in literature. In our study, we are concerned only with the leaf nodes, which comprise the set of possible faults in our network-centric application with specific modes for the TTE communication technology used such as vSN for incorrect sequence numbers in rate-constrained traffic.

Faults listed in Fig. 4 can be introduced in specific components, or can be detected and stopped or transformed to a different fault at some components and thus propagated downstream. A number of fault detection and protection mechanisms are builtin to the different network components and depend on the type of traffic mode: Time-Triggered (TT), Rate-Constrained (RC) or Best-Effort (BE). We model the fault propagation probabilistically, taking into account the component failure rates and the efficacy of the protection mechanisms. We highlight some of the fault detection and protection mechanisms in Table I. The TTE specification [15] contains more details.

IV. A Case Study: Brake-by-wire

Papadopoulos et al. [13] describe an initial brake-by-wire model that we are extending to include communication of signals from the wheel brakes to the brake lights as well as feedback to the driver from the light sensors about whether any of the bulbs need to be replaced. We also include a safety-critical communication of brake signals to the motor control logic in order to prevent opening the throttle while braking.

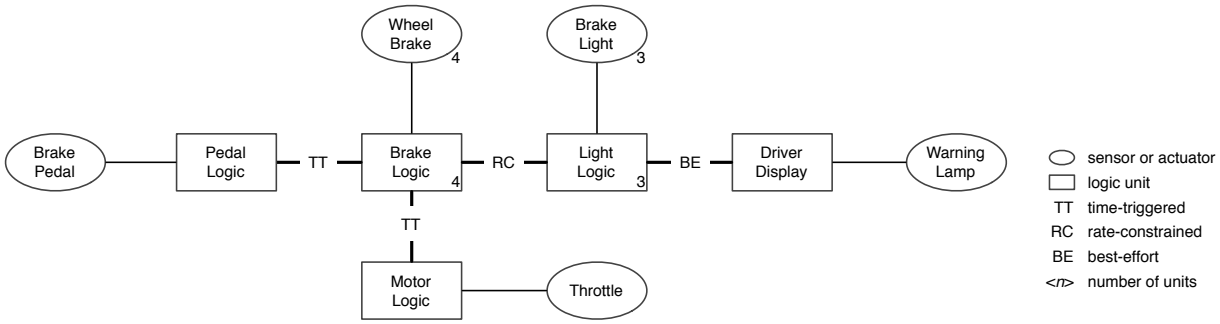


Fig. 5. High-level overview of brake-by-wire system design

TABLE I
FAULT DETECTION AND PROTECTION MECHANISMS

Mechanism	Traffic Type	Components
FCS/CRC addition	TT, RC, BE	ES, SW
VL ID & destination address	TT, RC, BE	ES, SW
Message length and type	TT, RC, BE	ES, SW
Payload length	TT, RC	ES, SW
Scheduled TT dispatch	TT	ES, SW
FIFO ordering	RC, BE	ES, SW
Traffic shaping	RC	ES
Timing window	TT	SW
BAG policy	RC	SW
Port BAG enforcement	BE	SW
Age check	RC	SW
Redundancy management	TT, RC	ES
Integrity checking	TT, RC	ES
Input exchange	TT, RC, BE	High Integrity
Output cross comparison	TT, RC, BE	High Integrity

TABLE II
TRAFFIC FLOWS IN BRAKE-BY-WIRE SYSTEM

VL	Sender	Receiver(s)	Type
1	Pedal Logic	Brake Logic 1, 2, 3, and 4	TT
2	Brake Logic 1	Motor Logic	TT
3	Brake Logic 2	Motor Logic	TT
4	Brake Logic 3	Motor Logic	TT
5	Brake Logic 4	Motor Logic	TT
6	Brake Logic 1	Light Logic 1, 2, and 3	RC
7	Brake Logic 2	Light Logic 1, 2, and 3	RC
8	Brake Logic 3	Light Logic 1, 2, and 3	RC
9	Brake Logic 4	Light Logic 1, 2, and 3	RC
10	Light Logic 1	Driver Display	BE
11	Light Logic 2	Driver Display	BE
12	Light Logic 3	Driver Display	BE

Fig. 5 shows a high-level overview of our brake-by-wire system design. Each sensor and actuator has a corresponding logic unit, which interfaces with the TTE communication infrastructure. Corresponding to the criticality of the signal, the TTE communication links are labeled as TT (high criticality), RC (medium criticality), and BE (low criticality). The highly critical paths should also be protected by redundancy. The following rules govern the expected behavior of the system.

- If brake pedal is engaged, brake at each wheel.
- If wheel brake is engaged, illuminate brake lights.
- If wheel brake is engaged, close throttle at motor.
- If brake light does not work, show warning.

From the general system specification, we model the system using the components and mechanisms introduced above. For each message flow, we assign a so-called *Virtual Link* (VL) between the source and the destination(s). During design refinement, each VL is mapped onto actual channels in the network architecture. Table II contains all 12 virtual links in our model of the brake-by-wire system.

V. MAUDE IMPLEMENTATION

A. Equational Logic and Maude

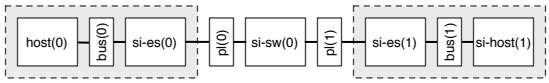
Equational logic (EL) [12] is the subset of first-order logic with $=$ as the only predicate symbol, and equations as the only formulas (i.e., there are no logical connectives). Despite being a very small subset of first-order logic, equational logic can be used to define any computable function. Furthermore, EL can be used as a programming language, by treating equations as left-to-right rewrite rules (i.e., ignoring the symmetry rule).

Maude is a multiparadigm executable specification language encompassing EL. The Maude interpreter provides efficient prototyping of quite complex test cases as well as built-in search and model checking capabilities.

For the fault analysis tool, we use (conditional) equations to specify fault propagation rules. Terms for the equations are built from operators and variables. Equational axioms are introduced with the keyword `eq` (or `ceq` for conditional equations) followed by the two terms being declared equal separated by the equality sign =.

B. Probabilistic Fault Analysis Using Maude

We have developed a network fault analysis in Maude that allows us to specify network topologies and traffic flows, and analyze the fault introduction and propagation in the network in a probabilistic way. The most important data structures in this framework are *network configurations*, consisting of a *network* and one or more *dataflows*. A simple network configuration specified in this framework is shown in Fig. 6.




```
op net : -> NetworkConfiguration .
eq net =
  (< si-host(0) | empty | out(0) >,
   < conn(0) | si-host(0) : out(0) to bus(0) : in(0) >,
   < bus(0) | in(0) | out(0) >,
   < conn(1) | bus(0) : out(0) to si-es(0) : in(0) >,
   < si-es(0) | in(0) | out(0),out(1) >,
   < conn(2) | si-es(0) : out(0) to pl(0) : in(0) >,
   < pl(0) | in(0) | out(0) >,
   < conn(3) | pl(0) : out(0) to si-sw(0) : in(0) >,
   < si-sw(0) | in(0) | out(0) >,
   < conn(4) | si-sw(0) : out(0) to pl(1) : in(0) >,
   < pl(1) | in(0) | out(0) >,
   < conn(5) | pl(1) : out(0) to si-es(1) : in(0) >,
   < si-es(1) | in(0),in(1) | out(0) >,
   < conn(6) | si-es(1) : out(0) to bus(1) : in(0) >,
   < bus(1) | in(0) | out(0) >,
   < conn(7) | bus(1) : out(0) to si-host(1) : in(0) >,
   < si-host(1) | in(0) | out(0) >,
   < conn(8) | si-host(1) : out(0) to null : in(0) >) ||
   df(0 | tt |
     faults(conn(0) | unknown ),
     faults(conn(1) | unknown ),
     faults(conn(2) | unknown ),
     faults(conn(3) | unknown ),
     faults(conn(4) | unknown ),
     faults(conn(5) | unknown ),
     faults(conn(6) | unknown ),
     faults(conn(7) | unknown ),
     faults(conn(8) | unknown )) .
```

Fig. 6. Example of a network configuration specified in Maude

The network part consists of a number of *network components*, each surrounded by angle brackets. The network specified here is an all-SI version of the network shown in Fig. 1. We have two SI hosts, two buses, two SI end systems, two physical links (PL), and an SI switch. Each component has ingoing and outgoing *ports*, connected by *connections*. Numbers are used as identifiers to distinguish different components of the same type. The bottom part of the network configuration is a dataflow (df). The dataflow specifies its traffic type (in this case `tt` for time triggered). It also lists all the connections that are part of the dataflow, with a *fault annotation* for each one. Before fault analysis, each fault annotation is `unknown`. The first objective of the fault analysis is to replace all the `unknown`s with boolean formulas representing the conditions under which the fault occurs.

Each network component can introduce and/or propagate faults of the different types shown in Fig. 4. The introduction/propagation behavior of each component is specified using one or more equations in the Maude specification. For example, the rule for transmitting SI host is shown in Fig. 7.

```
ceq
  (< SIHost | INS | Out1,OUTS >,
   < Conn1 | SIHost : Out1 to C : In1 >, Net) ||
   (df(DF | TType | faults(Conn1 | unknown), CS),DFS)
  =
   (< SIHost | INS | Out1,OUTS >,
    < Conn1 | SIHost : Out1 to C : In1 >, Net) ||
   (df(DF | TType |
      faults(Conn1 | om : OMout, com : COMout,
        vSA : VSAout, vDA : VDAout, vSN : VSNout,
        vLen : VLENout, vData : VDATAout,
        vFCS : VFCSout, te : TEout, tl : TLout ),CS),
        DFS)
if
  outgoingBusConn(< Conn1 | SIHost : Out1 to C : In1 >,
    Net, CS) /\
  HFail := pr(si-host-out-fail,SIHost : Out1) /\
  OMout := HFail /\
  COMout := HFail /\
  VSAout := if TType == be then HFail else false fi /\
  VDAout := HFail /\
  VSNout := false /\
  VLENout := if TType == be then HFail else false fi /\
  VDATAout := HFail /\
  VFCSout := false /\
  TEout := HFail /\
  TLout := HFail .
```

Fig. 7. Fault introduction rule for SI hosts

This is a conditional equation, consisting of a left side before the = sign, a right side after the = sign, and a condition after the `if`. Note that all the capitalized parts are variables. The rule will execute when any part of the network matches the left side, and the condition is true. Thus, this particular rule applies to SI hosts with an outgoing connection that is part of a dataflow. The result of executing the rule is to replace the matched part of the network with the right side of the equation. The only difference between the left and right sides of the equation is that in the right side, the `unknown` fault annotation has been replaced by a number of actual fault annotations for the different fault types (`om`, `com`, etc.). The variables that represent the boolean formulas (`OMout`, `COMout`, etc.) are defined in the condition part of the equation. The condition first creates a probabilistic variable (using the `pr` operator) called `HFail` of type `si-host-out-fail`. This variable represents a failure associated with the outgoing port of the host.[1] This variable is then used in the variable assignments that follow in a straightforward way. Most of the faults simply happen if and only if the host failure happens.

Because a transmitting host is the originator of any data it sends, unlike all other components, it cannot

[1]It is also possible to create more fine-grained failure types, rather than one big failure type for the whole component.

propagate faults, only introduce them. For a more typical case, Fig. 8 shows part of the condition of the rule for SI end systems. Here, we have both propagation and introduction of faults, and separate possible failures on the incoming and outgoing ports, respectively.

```
ESInFail := pr(si-es-in-fail,SIES : In2) /\
ESOutFail := pr(si-es-out-fail,SIES : Out2) /\
OMout := OMin or (COMin or  VDAin or TEin or TLin) and
  not ESInFail or ESInFail or  ESOutFail /\
COMout := ESOutFail /\
VSAout := ESOutFail /\
VDAout := (VDAin and ESInFail) or ESOutFail /\
VSNout := if TType == rc then ESOutFail else false fi /\
VLENout := ESOutFail /\
VDATAout := VDATAin or ESInFail or ESOutFail /\
VFCSout := ESOutFail /\
TEout := ESOutFail /\
TLout := ESOutFail
```

Fig. 8. Part of fault rule for SI end systems

Note that the incoming failures (`OMin`, `COMin`, etc) are themselves (possibly complex) formulas. Thus, the execution of the equations builds up these formulas in a combinatorial way. We are primarily interested in the fault annotation on the last connection of the dataflow, i.e., the combined effect of all of the network components that data has to go through. As an example, Fig. 9 shows the fault formula for `com` faults on connection 8 of our example network, after doing the fault analysis.

```
pr(si-es-out-fail, si-es(1) : out(0)) or
pr(si-es-in-fail, si-es(1) : in(0)) and
  (pr(si-sw-out-fail, si-sw(0) : out(0)) or
    pr(si-es-out-fail, si-es(0) : out(0)) and
  pr(si-sw-in-fail, si-sw(0) : in(0)))
```

Fig. 9. A fault formula generated by the Maude fault analysis

Again, the formula is a boolean combination of probabilistic variables that represent the occurence of failures in various components through which the data is transmitted. In general, these formulas can become quite large. The next step of fault analysis is to evaluate the formula. Given some actual numbers for each type of failure (`si-es-out-fail`, `si-sw-in-fail`, etc.), we want to know the probability for the whole compound formula. There are both exact and approximate ways of doing this. In the following, we present our first implementation of both types of methods.

To calculate the probability of one of our formulas, we use the following rules of probabilistic logic.[2]

$Pr(\texttt{not}\ P) = 1.0 - Pr(P)$
$Pr(\texttt{true}) = 1.0$
$Pr(\texttt{false}) = 0.0$
$Pr(P\ \texttt{and}\ Q) = Pr(P) * Pr(Q)$
if P and Q are independent
$Pr(P\ \texttt{or}\ Q) = Pr(P) + Pr(Q)$
if P and Q are disjoint

[2]http://en.wikipedia.org/wiki/Probability_space

The issue here that prevents a straightforward calculation is the side conditions for the and- and or- rules. The two subformulas are typically *not* independent or disjoint due to variables appearing in both. There are different approaches to computing with probabilistic logic [11].

One approach is to convert the whole formula into a form where all conjunctions are independent and all disjunctions are disjoint. It turns out that formulas in full disjunctive normal form (full DNF) are of the type described. A formula is in DNF if and only if it is a disjunction of one or more conjunctions of one or more literals (a literal is either a propositional variable or a negated variable). For example, $(P\ \texttt{and}\ Q)\ \texttt{or}\ (P\ \texttt{and}\ R)$ and $P\ \texttt{or}\ (Q\ \texttt{and not}\ R)$ are both in DNF. A formula is in *full* DNF if and only if it is in DNF and if each of its variables appears exactly once in every clause. Any formula can be converted to full DNF. For example, the first formula above becomes

$(P\ \texttt{and}\ Q\ \texttt{and}\ R)\ \texttt{or}\ (P\ \texttt{and}\ Q\ \texttt{and not}\ R)\ \texttt{or}$
$(P\ \texttt{and not}\ Q\ \texttt{and}\ R)$

Note that each formula has a unique full DNF form, but not a unique DNF form. The full DNF form can be interpreted as the entries in a truth table that make a formula true. For example, for the formula above (with 1 for true and 0 for false) the truth table is shown in Table III.

TABLE III
FULL DNF FORMULA AS A TRUTH TABLE

P	0	0	0	0	1	1	1	1
Q	0	0	1	1	0	0	1	1
R	0	1	0	1	0	1	0	1
(P and Q) or (P and R)	0	0	0	0	0	1	1	1

Each column describes one complete variable assignment, or "possible world." Each column differs in at least one position. Hence, each column is disjoint from all the others. They describe different states of affairs that cannot be true at the same time. Similarly, each row for the atomic variables is independent of the others, since we already stated that we assume that the atomic variables are independent.

Looking at the table, we see that the entire formula (last row) is true exactly in the situations described by the rightmost three columns. In other words, when P is true, Q is false, and R is true, OR when P is true, Q is true, and R is false, OR when all three are true. As can be seen from this verbiage, each column can be interpreted as a conjunctive formula, and the combination of several columns can be interpreted as

a disjunction. Each column contains all the variables. Hence, what we get from the table is a full DNF formula, the probability of which we can easily compute using the rules above. Our first implementation used this approach, within the Maude framework.

The problem with the "full DNF" method described above is its computational complexity. The conversion to full DNF (or even regular DNF) form is exponential in the number of variables contained in the formula. Thus, we have been able to use this method only for small examples, and as a reference implementation with which to compare other approaches. Our second approach uses Binary Decision Diagrams (BDDs) as a representation instead of converting the formulas to full DNF form. There are several highly efficient BDD packages. These can very quickly return all variable assignments that satisfy a given formula/BDD. Given those variable assignments, we can calculate our probabilities the same way as with the full DNF formulas. Our implementation uses Maude to generate the fault formulas, and then uses the JavaBDD[3] library to analyze the results in Java. In our tests, computation time with BDDs was negligible, even where the original implementation crashed or stalled.

It is still possible to hit upon limitations with the BDD approach, as its theoretical worst-case behavior is still NP-hard. Another approach to calculating the probabilities is to use an approximate, sampling-based method. The idea is to generate a number of samples, where each sample randomly determines the values (true or false) of all the probabilistic variables, according to their probabilities. The formula is then evaluated with all the variables replaced with true or false. The whole formula thus becomes true or false for each sample. With enough samples, the distribution of true/false for the entire formula will approach the correct result that would be calculated by an exact method such as the one described above. We have implemented a sampling-based method using a combination of Java code and Maude. The implementation is capable of evaluating several hundred thousand samples per second. However, our failure probabilities are sometimes on the order of 10^{-9}. In order to "catch" these tiny probabilities, we need a huge number of samples and hours of computation time. Thus, this method is still not fast enough for practical purposes. However, we are exploring alternative, weighted sampling methods that may be applicable, which would make the computation much faster.

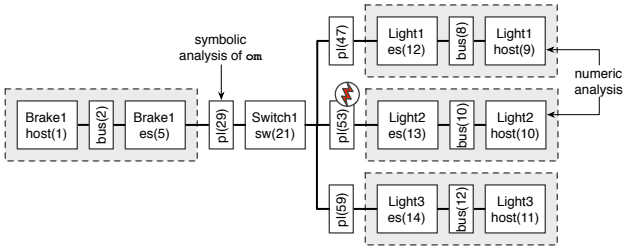


Fig. 10. Example VL 6 (rate-constrained, no path redundancy) with points of symbolic and numeric analysis and fault introduction

C. Fault Analysis Results for BBW

Let us reconsider the BBW application introduced in Section IV. Fig. 10 depicts the subgraph of VL 6 as an example. In our implementation, we model buses and physical links as part of the network, so as to define failures on such links. In VL 6, a signal from one wheel brake is sent to all three brake lights to illuminate. The numbers in parenthesis denote identifiers in the Maude code.

In Fig. 11 we show the symbolic analysis for omission faults for the connection from the Brake1 end system to the physical link toward Switch1. The symbolic analysis for connections further downstream becomes too large to print here.

```
pr(bus-fail, bus(2)) or
pr(si-host-out-fail, si-host(1) : out(0)) or
pr(si-es-in-fail, si-es(5) : in(0)) or
pr(si-es-out-fail, si-es(5) : out(1)) or
not pr(si-es-in-fail, si-es(5) : in(0)) and
 (pr(bus-fail, bus(2)) or
  pr(si-host-out-fail, si-host(1) : out(0)))
```

Fig. 11. Results of VL 6 symbolic analysis (for connection from Brake1 to physical link with Switch1)

TABLE IV
EXAMPLE FAILURE PROBABILITIES FOR BBW

Fault Type	Probability*)
FCS check	10^{-9}
physical link	10^{-9}
bus	10^{-9}
SI host input	10^{-6}
SI host output	10^{-6}
SI end system input	10^{-6}
SI end system output	10^{-6}
SI switch input	10^{-6}
SI switch output	10^{-6}

*) These are not representative numbers for real network components.

When analyzing the quantitative fault propagation of a VL, we first supply the failure probabilities for each

[3]http://javabdd.sourceforge.net/

component type as a valuation set. Table IV shows the failure probabilities used for the example computation below. Fig. 12 contains the result of analyzing VL 6 at the receiving hosts Light1 (without any introduced fault) and at Light2 (with a physical link failure introduced between Switch1 and Light2).

```
om : 7.003789723448e-5,     om : 1.0,
com : 1.00001999971e-5,     com : 1.00001999971e-5,
vSA : 1.001000000009e-5,    vSA : 1.001999979992e-5,
vDA : 1.001010010009e-5,    vDA : 2.001989959991e-5,
vSN : 3.002969910071e-5,    vSN : 1.0,
vLen : 1.001010010009e-5,   vLen : 2.001989959991e-5,
vData : 1.001010010211e-5,  vData : 2.001989959991e-5,
vFCS : 1.0010000000099e-5,  vFCS : 1.001999979992e-5,
te : 1.000000000002e-10,    te : 1.000000000002e-10,
tl : 1.999990000002e-10     tl : 1.999990000002e-10
```

Fig. 12. Results of VL 6 analysis (without and with introduced fault) using scientific notation (i.e., 1e-9 $= 10^{-9}$)

One can see clearly that a faulty physical link at the input of Light2 causes an omission error and also a violation of the sequence numbers that are expected during a rate-constrained communication. If the traffic mode were to be set to time-triggered, the sequence numbers no longer apply. However, the omission error would still occur. If the system-level requirements call for better reliability of the message delivery from the brakes to the lights, the system design engineer could now test an architecture with path redundancy or other enhancements.

VI. Conclusion and Future Direction

The Maude-based Fault Analysis tool provides a means to evaluate the effectiveness of fault protection mechanisms in various network architectures. We have implemented the tool and applied it to various sample networks. A special-purpose editor automatically generates input to the Maude Fault Analysis tool. Because of space limitations, we have not featured the editor in this paper. However, the network editor and the fault analysis tool will be made publicly available.[4]

The evaluation on a simple brake-by-wire network with high- and standard integrity components, and path redundancy but no system redundancy, illustrates the complexity of fault formulas. The analysis tool successfully performs symbolic and quantitative fault analysis.

Future extensions concern extending the underlying network model and fault propagation rules for system redundancy schemes and analyzing the symbolic fault formulas to determine main contributors to faults, so as to point the designer where additional protection would have the most payoff. Another direction includes modeling fault propagation with temporal aspects, possibly extracting dynamic fault trees for better comparison with other approaches. An additional temporal aspect to consider is how failure rates change over time (e.g., as a "bathtub curve"). Our current tool assumes constant failure rates.

[4]http://promise.csl.sri.com

References

[1] ARINC Report 664P7-1. *Aircraft Data Network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network*, Sept. 2009.

[2] R. W. Butler. A primer on architectural level fault tolerance. Technical Report TM-2008-215108, NASA, Feb 2008.

[3] M. Chérèque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron. Active replication in delta-4. In *FTCS*, pages 28–37, 1992.

[4] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[5] X. Ge, R. Paige, and J. McDermid. Probabilistic failure propagation and transformation analysis. In *Computer Safety, Reliability, and Security (SAFECOMP)*, volume 5775 of *Lecture Notes in Computer Science*, pages 215–228. 2009.

[6] X. Ge, R. Paige, and J. McDermid. Analysing system failure behaviours with PRISM. In *International Conference on Secure Software Integration and Reliability Improvement Companion*, pages 130–136, June 2010.

[7] R. C. Hammett and P. S. Babcock. Achieving 10^{-9} dependability with drive-by-wire systems. In *SAE 2003 World Congress and Exhibition, March 2003,Session: Safety Critical Systems*, Detroit, March 2003.

[8] B. Kaiser, C. Gramlich, and M. Förster. State/event fault trees – a safety analysis model for software-controlled systems. *Reliability Engineering & System Safety*, 92(11):1521–1537, 2007.

[9] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidurai. The maft architecture for distributed fault tolerance. *IEEE Transactions on Computers*, 37:398–405, 1988.

[10] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1997.

[11] N. J. Nilsson. Probabilistic logic. *Artif. Intell.*, 28(1):71–87, 1986.

[12] M. J. O'Donnell. Equational logic as a programming language. In R. Parikh, editor, *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, page 255. Springer, 1985.

[13] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering & System Safety*, 71(3):229–247, Mar. 2001.

[14] Y. Papadopoulos and J. A. McDermid. Hierarchically performed hazard origin and propagation studies. In *Computer Safety, Reliability and Security*, volume 1698 of *Lecture Notes in Computer Science*, pages 688–688. 1999.

[15] W. Steiner. *TTEthernet Specification*. TTA Group, 2008. Available at http://www.ttagroup.org.

[16] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch. TTEthernet: Time-Triggered Ethernet. In R. Obermaisser, editor, *Time-Triggered Communication*. CRC Press, Aug 2011.

[17] Y. Yeh. Triple-Triple Redundant 777 Primary Flight Computer. In *IEEE Aerospace Applications Conference*, pages 293–307. IEEE, 1996.

Integrating Sleep Scheduling and Compressed Sensing in Sensor Networks

Debojit Dhar and Sathish Gopalakrishnan
Department of Electrical & Computer Engineering
The University of British Columbia

***Abstract*— We consider the interaction between the natural redundancy of sensor data for a variety of data gathering applications and the need for reducing energy consumption when such data is obtained through low-energy nodes communicating wirelessly. We employ compressed sensing as a mechanism for reducing the number of sensor nodes that need to be active at any given point in time for effective data reconstruction. Correspondingly we describe a decentralized coordination scheme that permits nodes to adjust their sleep and activity schedules subject to connectivity and coverage requirements that allow for satisfactory data reconstruction fidelity. Such problems are intractable and our schemes are approximation algorithms for minimizing the number of active nodes while adhering to the mentioned requirements. Our research contributions are two-fold: (i) we derive bounds on the performance of our coordination scheme and (ii) we evaluate the effectiveness of integrating compressed sensing with our coordination scheme through simulations based on a data generation tool that models some physical phenomena.**

I. Introduction

Wireless sensor networks are an ideal platform for monitoring and controlling physical environments. In such networks, nodes communicate with each other and convey data periodically to a base station or fusion centre over multi-hop wireless links. A first-order constraint for some applications is the energy budget because the network may be deployed in harsh, and not easily accessed, terrain. Application users would then prefer to maximize the interval between returning to the sensing field to replace batteries.

The lifetime of a wireless sensor network can be maximized in several ways. One obvious solution is to add more batteries to sensor nodes but this approach leads to an increase in the form factor of the nodes, which makes them unsuitable for several applications. An alternative solution is to utilize a high density of nodes and employ only a fraction of these nodes at any given time instant. This approach requires that we determine appropriate duty cycles for different nodes. This approach is also well suited to the case when nodes may be equipped with rechargeable batteries and the recharge operation (whether through solar energy or other harvesting methods) requires nodes to transition to an inactive state. A further advantage of high-density deployments is fault tolerance because the redundancy present in the system enables correct functioning even when certain nodes fail. As a consequence of the advantages and technological feasibility of high-density sensor networks, we will discuss methods that are well suited to lifetime management in this scenario.

In particular, we determine near-optimal methods for duty cycling between sensor nodes so as to preserve coverage of the sensing field and connectivity between active nodes and the base station. Decentralized operation of sensor nodes is desirable because this limits the effect of failures. We therefore focus on decentralized methods for ensuring coverage and connectivity.

In certain applications where the data observed by different sensors is correlated, as is the case when extracting thermal profiles of a region, one can utilize a limited number of nodes and yet reconstruct the complete profile through *compressive sensing* [14]. Thus, when data observed by sensor nodes is correlated, duty cycling does not diminish the accuracy of data gathering. We will emphasize the use of compressive sensing when appropriate to extend network lifetime while acquiring data with acceptable error bounds. An advantage of compressive sensing is that we can keep only a few nodes active at every time instant even when node density is not high if the data correlation is high. It is important to note that even when compressive sensing is employed, we do need methods to ensure appropriate coverage and a connected backbone to gather the sensed data.

Whereas the idea of duty cycling to save energy is not new, our central contributions are to:

- develop a decentralized, and near-optimal, method for duty cycling while preserving connectivity and coverage in the sensing field (Section IV),
- establish bounds on the performance of the node activation (sleep scheduling) algorithm (Section IV-B), and
- utilize compressive sensing for reconstructing missing data in the appropriate scenarios (Section V).

When describing our technique for selecting nodes to be active we also establish the hardness of obtaining optimal solutions to that problem. Our methods extend the state of the art in sensor sleep scheduling (or duty cycling). In designing our techniques, we were guided by the need for simplicity because implementation of algorithms for WSNs are often more challenging than they may appear [34].

Through simulation, we are able to demonstrate that we can extend the lifetime of certain networks by a factor of 5 while ensuring that data quality is sufficiently high. While our methods are useful for a variety of sensor network applications, they are not tailored for some situations such as intruder detection and object tracking [27] that require fast response times and where data correlation between sensors is poor.

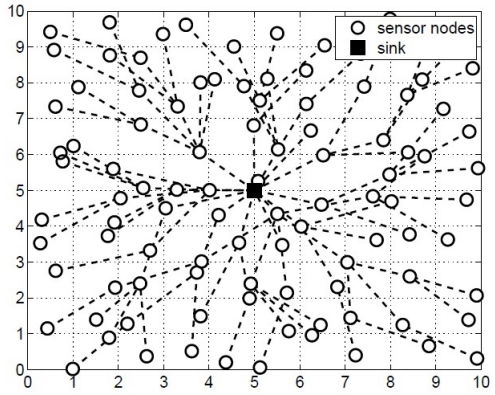


Fig. 1. Sensor network with active nodes connected to a base station.

II. System model and assumptions

In our model of a wireless sensor network, we consider a set of sensor nodes that can communicate with each other wirelessly. We assume that each sensor node knows is location, either through GPS information or through other well-studied techniques such as localization [15]. We make no assumptions about the topology of the network. In other words, nodes could be distributed according to a regular pattern such as a grid or be distributed randomly over the sensing region.

Each node can communicate with other nodes that are within its transmission region. Nodes within the transmission region of a node are called its *neighbours*. Each node also has a sensing range, which determines the area that it is able to gather data from.

In the network, a particular node is assumed to be the base station or data sink. This node does not have energy constraints because it may be connected to a permanent power source. The base station is computationally more powerful than other nodes in the network. This node is, however, subject to the same communication constraints as other nodes in the network. Henceforth, we will use the term *node* to refer to nodes that are not the base station, and we will use the term data sink or base station when we explicitly refer to this particular node.

We assume that nodes can be in one of two states: active or inactive. In the active state, a node senses data and communicates with a subset of its neighbours to ensure that the data is transmitted to the base station, possibly over multiple hops. We do not assume that intermediate nodes fuse data to reduce the size of data transmitted although data fusion of this nature can be accommodated by the methods that we propose.

We assume that the network is connected if there is a path from every active node to the base station, and all nodes on the path are also active. The network provides full coverage if the sensing ranges of all active nodes covers the entire sensing region. We do make the assumption that the initial deployment of nodes provides complete coverage when all nodes are active.

III. Related work

Prior work on data gathering techniques for sensor networks have explored various avenues [20], [21] to reduce the cost of acquiring data from a sensor network. These include energy-efficient routing protocols [28], aggregation techniques [19], [10] and data store abstractions [21], [20]. These schemes did not adopt duty cycling but focused on reducing the energy cost of communication, and the gains achieved were limited because application-aware duty cycling can explicitly transition nodes to an inactive state and achieve greater energy savings. Researchers have also developed policies that use transmission power regulation to provide energy savings [16]. These policies require complicated MAC and routing protocols. Since our aim is to build a system which is simple to implement, we try to avoid the use of such complex protocols.

Sleep scheduling and duty cycling have been proposed in the past, and the main goal is to preserve some level of connectivity and coverage among nodes that are active. Whereas prior work has demonstrated the value of exploiting low-power sleep states in sensor nodes [32], we have developed a decentralized scheme for sleep scheduling that is configurable in terms of the active neighbourhood size for any node. We also establish performance bounds on our scheme. We gain knowledge from the previous body of work on connectivity and coverage in sensor networks. While some papers stress coverage [29], [5], [33], others stress connectivity [6], [9], [36]. Most of the coverage techniques try to find a relation between the sensing range and communication range of the nodes. Such results are relevant to our proof for bounding the sub-optimality of the CC Algorithm.

We employ compressed sensing when possible to reconstruct missing data in a duty-cycled WSN. Compressed sensing has been applied in image processing [13], [31] for recovering dead and unreadable pixels from surrounding pixels. Jarvis et al. [14] were the first to utilize this technique for sensor networks. Jarvis et al. used compressed sensing to conserve bandwidth in a WSN and did not tackle the problem of lifetime maximization.

Other work connecting sensor networks to compressed sensing [1], [2], [24], [22] has been directed at devising better reconstruction algorithms and have tried applying compressed sensing to solve different shortcomings in sensor networks. Quer et al. [23] tested these algorithms on real workloads but even they did not articulate concrete implementation metrics and did not measure energy savings. Other researchers have come up with compression schemes that can be adapted efficiently with routing [12]; our work is different from them in the sense that compression and routing are dealt with independently. We present an algorithm that constructs a connected set of active nodes and we assume tree-structured communication [30] to gather data at the data sink.

IV. Maintaining connectivity and coverage

Our initial goal is to ensure that, in a dense deployment of sensor nodes, a sufficient number of nodes remain in the active state at any time instant to ensure sensing coverage and network connectivity. Other nodes in the network can transition to the inactive state and conserve energy. Apart from requiring that nodes know their location (see Model), we assume that nodes employ some light-weight time synchronization mechanism. We divide time into discrete activity intervals and each node, according to a decentralized decision-making policy, decides to be

active or inactive in that interval. Only active nodes can participate in sensing, processing, and communication. A node can communicate only with neighbours that are also active.

The network has a distinct base station and all nodes should be able to send data to the base station. The base station is assumed to be active during all activity intervals. An active node should either be a neighbour of the base station or should have at least one active neighbour that has a path to the base station.

To ensure sufficient coverage of the sensor field, we also require that every node (active or inactive) have at least one neighbouring node that is active in each time interval.

We generalize the connectivity and coverage requirements through a parameter γ. γ is the number of nodes that must be active in the neighbourhood of any sensor node v. $\gamma = 1$ provides basic connectivity and coverage while $\gamma > 1$ is useful when more robust coverage and fault tolerance is required. The value of γ has a direct relationship with the fraction of nodes we want to be kept active at any time instant. In section IV-G, we will discuss more about the methods for deriving γ.

The properties that should be satisfied by the decentralized activity planning algorithm can now be listed:

- Each node v with γ_v neighbours must have at least $\min(\gamma, \gamma_v)$ active neighbours at any time instant. When deployments are dense $\gamma_v \gg \gamma$.
- The algorithm will guarantee node coverage – all active nodes will be connected and every node (active or inactive) will have an active neighbour.
- To ensure load balancing among nodes, the set of active nodes will change from one activity interval to the next.

A. Decentralized activity planning

We now present a simple randomized scheme that preserves the properties we desire. Randomization ensures that the set of active nodes changes between activity intervals. The following steps are executed at each node v in the network. Our scheme requires that every node v transitions to the active state at the end of every activity interval, negotiates with its neighbours and then determines its state for the next activity interval.

The following steps are followed at every node v:

1) Pick a random ballot b_v where b_v is a real number in the set $[0,1]$.
2) Broadcast b_v and receive the ballots from neighbours. The neighbours are denoted by the set γ_v. Let the set of ballots received be B_v.
3) Broadcast B_v and receive B_w from each $w \in \gamma_v$.
4) If $|\gamma_v| < \gamma$ or $|\gamma_w| < \gamma$ for any $w \in \gamma_v$, then stay active and terminate the planning phase.
5) Compute $A_v = \{w | w \in \gamma_v \text{ and } b_w < b_v\}$.
6) Transition to the inactive state if the following two conditions are true, else stay active.
   - Any two nodes in A_v are connected either directly themselves or indirectly through some node w within v's two-hop neighbourhood such that $b_w < b_v$.
   - Any node in γ_v has at least γ neighbours in their A_v.

We shall refer to the algorithm just described as Algorithm CC (connectivity and coverage). We shall now discuss the correctness of this algorithm and then derive bounds on the sub-optimality of the algorithm.

B. Correctness

Theorem 1: Algorithm CC ensures that any node v has at least $\min\{\gamma, \gamma_v\}$ active neighbours.

Proof: When $\gamma_v < \gamma$, then none of v's neighbours will transition to the inactive state. When $\gamma_v \geq \gamma$ then we shall establish the result by contradiction. Assume that for $i \leq \gamma$ the neighbour of v with the ith lowest ballot transitions to the inactive state. Call this neighbour w. A_w will have at most $i-1$ nodes that are neighbours of v. Since $i-1 < \gamma$, w cannot transition to the inactive state thus resulting in a contradiction. ■

Theorem 2: If the original graph was connected then Algorithm CC results in a connected graph.

Proof: We shall establish this result by contradiction. Suppose that the output graph is disconnected. Add the deleted nodes back in the graph in ascending order of their ballots, and let v be the first node that makes the graph connected. By the time we add v, all members of A_v are already present. Moreover, nodes in A_v are already connected since they are connected by nodes with ballots at most b_v. Let w be a node that was disconnected from A_v but gets connected to A_v by v. This contradicts the rule that v can sleep only if all its neighbours (including w) are connected to at least γ nodes in A_v. ■

C. Bounding the sub-optimality of algorithm CC

The problem of maintaining coverage and connectivity in a graph (network) even for the special case of $\gamma = 1$ is NP-complete. This is seen by noting that the problem when $\gamma = 1$ reduces to the minimum dominating set problem that is NP-complete. A natural question to answer then concerns bounds on the sub-optimality of the proposed algorithm. More specifically, if the number of nodes kept active by an optimal algorithm is a_{opt}, we would like to know how far a_{cc}, the number of nodes kept active by Algorithm CC, is from a_{opt}.

Obtaining a bound on the sub-optimality is difficult in the general setting but we will study the particular setting when all nodes can directly communicate with all nodes that are within a surrounding disc of radius r and when node density is sufficiently high.

Theorem 3: For any $\gamma \geq 1$, suppose n nodes are placed, uniformly at random, within a region such that each node has (on average) at least $4(\gamma + \log n)$ neighbours. Then, with high probability, $a_{cc} = O(\log n) a_{opt}$.

Proof: We shall find a lower bound on a_{opt} and an upper bound on a_{cc} so as to estimate the sub-optimality of Algorithm CC.

Let G be the graph of all nodes and let $\nu = 4(\gamma + \log n)$ be the average node degree in G. Applying Chernoff bounds, with high probability, $\nu/4 \leq \gamma_v \leq 4\nu$.

Let G_{opt} be the graph that is produced by an optimal algorithm. This graph contains all nodes in G but all edges involving two inactive nodes are deleted. In G_{opt}, each node has at least γ neighbours and therefore the number of edges

in G_{opt} is at least $n\gamma/2$. Further, since nodes in G_{opt} have at most $4v$ neighbours, the total number of edges is at most $4v\,a_{opt}$. Therefore, we have

$$4v\,a_{opt} \geq n\gamma/2 \Longrightarrow a_{opt} \geq n\gamma/(8v).$$

Consider $j = C\gamma n \log n/v$ for some appropriate constant C and run Algorithm CC on graph G. Let b be the jth smallest ballot selected by a node in G. We apply a lemma (Lemma 1) and note that all nodes that drew a ballot greater than b are inactive w.h.p.. It is possible that some nodes will ballots less than b_j may also be inactive but this claim is sufficient to establish a bound on a_{cc}, w.h.p.

By this claim, we have $a_{cc} \leq (C\gamma n \log n)/v = (8C\log n)\cdot n\gamma/(8v) \leq (8C\log n)a_{opt}$. Therefore, $a_{cc} = O(\log n)a_{opt}$, w.h.p.. ■

We shall now state and prove the lemma that we applied in the proof of the sub-optimality theorem.

Lemma 1: When we run Algorithm CC on a graph G, for some constant C, let $j = C\gamma n \log n/v$ and let b be the jth smallest ballot drawn by a node in G. W.h.p., all nodes with ballots greater than b are inactive.

Proof: We can view the process of constructing G as one of first selecting node ballots and then placing the nodes at random. Let v be a node with a ballot that is not in the bottom j ballots. Let D_v be a disc of radius r around v. The average node degree is v, and therefore we have that each node is placed in D_v with probability v/n. Let A_v be the active nodes around v and let A'_v be the nodes in A_v with ballots in the top j ballots. Then, $|A'_v|$ is distributed according to a binomial$(j, v/n)$ distribution. Applying Chernoff bounds, w.h.p., there are at least $x = jv/(4n) = (C\gamma \log n)/4$ randomly placed nodes in D_v that have ballots less than b (and hence have ballots less than b_v).

Consider any two nodes u and w in A_v, and let D_u and D_w, respectively, be discs of radius r around these two nodes. Define $D_{v/2}$ to be the disc of radius $r/2$ around v; two nodes in $D_{v/2}$ will be neighbours. Because u and w are in D_v, the areas of overlap between D_u and $D_{v/2}$ and between D_w and $D_{v/2}$ are each at least 1/12 of the area of D_v. Using the fact that $|A'_v| \geq x$ is logarithmic size, at least one node u' (w') lies within the intersection of D_u and $D_{v/2}$ (D_w and $D_{v/2}$). Additionally, u and w are connected by the path $u \leftrightarrow u' \leftrightarrow w' \leftrightarrow w$. The expected number of nodes in A'_v that fall in the intersection of D_u and D_v is at least $x/3$. Hence, using Chernoff bounds again, w.h.p. any node u in N_v has at least γ neighbours from A'_v. Thus, w.h.p., all the conditions for node v to transition to the inactive state are satisfied. ■

D. Energy optimizations

The decentralized activity planning method we have proposed is effective in providing coverage and connectivity in a sensor network deployment. It however requires local communication at every decision epoch. Every cycle, a node has to communicate its ballot value b_v and the set of received ballots B_v to all its one-hop neighbours γ_v. In order to restrict this overhead, every node v should be provided with:

1) The set of random number generator seeds R_v that is used to generate ballots b_v for all its one-hop neighbours γ_v.
2) The set of random number generators R_w from each $w \in \gamma_v$.

This indicates that every cycle, each node can calculate b_w and B_w for all its neighbours which relieves it from having to communicate the ballot values. Since computation is a much cheaper operation than communication [4], we end up being energy conservant.

E. Sleep scheduling in the communication disc model

A further optimization in Algorithm CC IV-C can be achieved in case the sensor nodes are placed along plain grids. In such regular distributions, it is easy to divide the area into smaller sub-grids and try to maintain connectivity and coverage in each sub-grid. Since we are now applying our optimizations in smaller sections of known dimension and density, better results can be obtained.

In this optimization method, we divide the entire sensed region into smaller sectors. The sector size depends on the communication range of the nodes and are designed such that all nodes in the sector would be a one or two hop neighbour to each other. This can be easily ensured using the communication disc logic where each node v has a communication disc D_v. Due of the regular grid pattern, we can ensure a certain number of nodes to be present in each node's D_v and hence decide on a sector size that asserts our assumption. Since our algorithm maintains coverage and connectivity in each sector, adjacent sectors can be assumed to be connected through one or more nodes. This makes it possible for each sector to act as an autonomous group and optimize behavior within itself. The connectivity and coverage algorithm inside each sector is similar to the decentralized activity planning algorithm with a few simplifications. Decision for the number of nodes to be kept awake in a sector is made based on the value of γ as it was done in the previous algorithm. For each node in the sector, there must be at least γ active neighbours.

In this model, each node v

1) Is provided a random number generator seed r_v which is used to generate its ballot b_v.
2) Keeps track of the seed of all other nodes in its sector, R_v.
3) Calculates the set B_w for each $w \in \gamma_v$.
4) If $|\gamma_v| < \gamma$ or $|\gamma_w| < \gamma$ for any $w \in \gamma_v$, then stay active and terminate the planning phase.
5) Remaining steps are the same as in the CC algorithm.

This distributed algorithm does not involve message exchange between the nodes and hence remains energy efficient. After certain intervals, which are usually long, the nodes exchange heartbeat messages to sync their seed information with other nodes in the same sector.

F. Simple coverage scheme

The previous schemes are valuable in case strict coverage and connectivity between the nodes is necessary. However, if strict coverage is not required, a simpler scheme might prove useful. We call this our *Simple Coverage Scheme*. In this scheme, each node v:

1) Is provided with a random number generator seed r_v, which is used to generate its random ballot b_v (a positive integer in this case).

2) Is provided with a set of random number seeds from all other nodes in its sector, R_v. (Similar to the sub-grid optimization, the nodes are divided into sectors.)
3) Is provided with λ, $\lambda = 1/f$ where f is the fraction of nodes desired to be active for the application. Once a node generates its ballot b_v, it performs modulo operation with λ to decide whether to wake up or not. (For example, if $b_v \ mod \ \lambda = 1$, wake up.)
4) If $b_v \ mod \ \lambda \neq 1$,
   - Compute $A_v = \{w | w \in \gamma_v$ and $r_v \ mod \ \lambda = 1\}$.
   - Compute $C_v = \{w | w \in A_v$ and $|A_w| = 0\}$.
   - If $|C_v| = 0$, then sleep.
   - If $|C_v| > 0$, for each w in C_v check all possible neighbours in the sector that can wake up to become a neighbour of w. If b_v is the lowest among all neighbours which are not a part of A_v, wake up, else go to sleep.

The Simple Coverage Scheme reduces computation for active nodes in each cycle compared to the sub-grid optimization. It also involves minimum communication, similar to the sub-grid optimization. However, this algorithm can only guarantee basic connectivity, as it does not ensure nodes with multiple active neighbours. Coverage is maintained through randomization.

G. Finding the value of γ

While using the decentralized connectivity and coverage algorithms, we based our assumption on the fact that the system has knowledge about γ, the number of active neighbours for each sensor node. However, calculating the value of γ is complicated and requires knowledge of several parameters. The value of γ is critical to the amount of energy savings, the accuracy and robustness of the sensor network deployment. To decide on a value of γ, we provide two simple strategies. Depending on the information available during deployment, we can use one of the following techniques to judge and distribute a value of γ to all the sensor nodes.

We present detailed description of the strategies that can be used to calculate γ.

The average neighbour strategy: In this method, we need knowledge of the percentage of nodes that are required to stay active for every cycle. This fraction value f is generally derived from *compressed sensing* V or calculated from the intended lifetime of the sensor network application. In this strategy, we must derive the average number of neighbours for a node in the network. This can be done at the set-up phase, if all nodes broadcast their one-hop neighbour count γ_v to the sink. The average number of neighbours, γ_{avg} can be calculated from the values of γ_v transmitted to the sink. γ can then be calculated as: $\gamma = |f \times \gamma_{avg}|$. Generally, in a deployment which provides a duty-cycling option, we can expect that the value of $\gamma \ll \gamma_{avg}$.

The fraction neighbour strategy: This is similar to the average neighbour strategy. We must derive the value of f as was done in the previous method. This value of f must then be distributed to all nodes in the network. Each node calculates its own γ from its neighbour count γ_v as: $\gamma = |f \times \gamma_v|$. An important outcome of this scheme is that it results in different values of γ for different nodes in the network. This is contradictory to our connectivity strategy which assumes that the value of γ is fixed. However the CC algorithm can easily accommodate variable values of γ. All steps in the CC Algorithm remain the same except for the 2nd part of the last step (6) where each node needs to check if each of its neighbour have their individual γ number of neighbours awake for a particular cycle.

V. Compressed sensing basics

Having discussed how we can maintain connectivity and coverage in conjunction with sleep scheduling, we now consider the use of compressed sensing for data reconstruction in certain sensor network applications. The advantage of compressed sensing is that we can reconstruct the reading of the inactive nodes in the sensing field with high accuracy. This allows us to deactivate a larger number of nodes while duty cycling and extract higher energy saving. Compressed sensing [11] relates to the problem of solving a system of under-determined linear equations where the solution is supposed to be sparse, i.e., $y = Ax$, where x is a sparse signal.

In the above equation, y is a compressed version of x. x can be considered to be a one dimension vector of length N while A is a matrix of size $M \times N$, with $M < N$. In general, an under-determined system has infinitely many solutions but it has been shown that if the solution itself is sparse, it is typically unique. Given enough time and computational resource, it is possible to solve the said problem by a brute force search. Mathematically it is represented as:

$min||x||_0$ *subject to* $y = Ax$

Unfortunately this is an NP hard problem [11]. However, solving the inverse problem by this method requires lesser number of equations. Compressed Sensing has proved that the said inverse problem can be solved by the following convex relaxation of the NP hard problem,

$min||x||_1$ *subject to* $y = Ax$

The l_1-norm is the tightest convex envelope of the NP-hard l_0-norm. This optimization problem can be solved by linear programming. It has been shown that intermediate between the NP hard (l_0-norm) problem and its convex relaxation (l_1-norm) lies the following non-convex relaxation [8],

$min||x||_p$ *subject to* $y = Ax$, $0 < p < 1$

It is possible to solve the above non-convex problem and the number of equations required to solve it is intermediate between the NP hard (l_0-norm) and the convex (l_1-norm) problem. The only issue theoreticians arise is that the non-convex problem may not converge to a global minima. However, practically this had never been the problem and this sort of non-convex methods give much better results than their more popular convex counterparts. Our evaluation results confirm this fact.

Practically, the actual signal is never sparse, but has a sparse representation in the transformed domain (Fourier, DCT, Wavelet etc.). In such a scenario, the inverse problem can be framed as,

$y = AS'z$

where $z = Sx$, $x = S'z$, S being the sparsifying transform. Therefore the corresponding optimization problem is:

$min||z||_1$ *subject to* $y = AS'z$

This formulation has been the basis for nearly all compressed sensing applications in signal processing.

From the data acquisition perspective, the reading from all the N nodes in the network is represented by x. The sink will receive only readings from the sensors that are active, which is the vector y. Since the sink has knowledge about the nodes that has transmitted, it would have information of the transformation matrices and hence can compute the original matrix x from the transmitted values.

VI. Experiments

A. Evaluation of connectivity algorithms

In order to test the coverage and connectivity algorithms, we built a simple simulator. The simulator accepts the co-ordinates of all the nodes in the network and the desired fraction of awake nodes as inputs. To test the CC Algorithm, we select a random placement of nodes over a sensing region. For the sub-grid optimization and the Simple Coverage Scheme, we consider a regular grid structure. We consider *average neighbour strategy* for neighbour assignment.

We ran experiments to verify effectiveness of the *Simple Coverage Scheme*. The simulator first outputs the initially active nodes and then the nodes that needed to wake up in order to maintain connectivity (Fig. 2).

We also conducted tests to compare the performance of our different connectivity and coverage algorithms. We found it difficult to compare our scheme with other connectivity algorithms since most of them have different objectives. Different objectives arise from the fact that the parameters we provide to our algorithm is unique and different from any other algorithm. To provide a simple comparison, we considered two different flavours of our algorithm – the CC Algorithm and the Simple Coverage Scheme. Each of them accept similar inputs but work differently. Such a comparison help users to decide which algorithm they want to implement.

In order to compare performance, we choose the number of awake nodes produced by each algorithm at the end of a cycle as the metric. For the same input parameters, the algorithm producing the lower number of nodes to maintain connectivity is an obvious winner. However, it is not clear which algorithm performs better as we can see from our experiments with the test data (Fig. 3). It seems from the results that the *Simple Coverage Scheme* generally performs better than its counterpart when the number of neighbour is high. It is upto us to decide the algorithm to be used. However, depending on the topology of the network and the required sparsity, a conscious decision can be made based on our results.

We also tested the bounds of the CC algorithm at different concentration of nodes in the sensing region. The same experiment would also establish the fact that the size and density of the network would stress the performance of our CC algorithm. In order to test our algorithm, we considered a 256 by 256 grid and varied the density or number of nodes in the area. For each node density, we varied the value of γ to see how the CC algorithm would react to varying application demands. As expected, the results (Fig. 4) conform to an increased number of awake nodes as the deployment becomes sparser. The number of neighbours for a given node, γ_v decreases with sparsity and we see that from some point, all nodes need to stay awake to support the value of γ set for that network. We also illustrate that the fraction of nodes that are awake also increases as γ is increased.

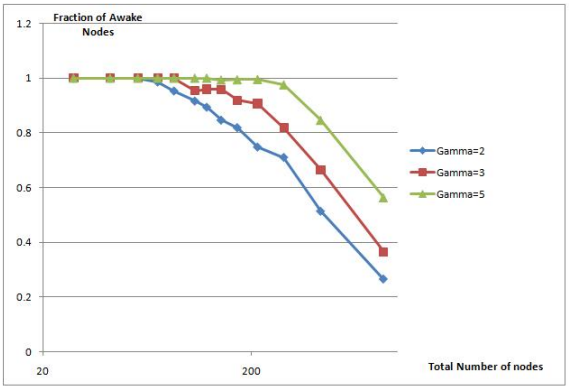


Fig. 4. Fraction of total nodes vs total number of nodes at different values of γ

1) Connectivity test: To prove that the connectivity algorithm is effective in reducing discontinuities in the network, we added a module to our simulator to test connectivity. In this test, we consider the same set of nodes that was used to illustrate the connectivity and coverage algorithm. At the end of a given cycle, we randomly pick nodes and test if there is a path to the sink from this node. The algorithm works in accordance to the following steps:

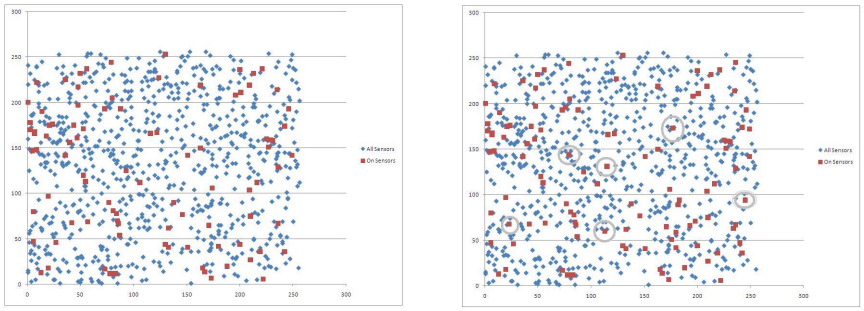
- For the randomly chosen node, find all awake neighbours in transmission range. Let the awake neighbours for node v be represented by a set A_v.
- For all $w, w \in A_v$, calculate their distance d_w from the sink S.
- Choose a neighbour node w_s such that for all $w, w \in A_v, d_w$ is the minimum.
- Replace v with w_s
- Continue till v or w_s is a neighbour of the sink, S.
- If there is a break in any of the above steps, connectivity is not maintained.

We ran exhaustive tests on the nodes to prove that all our connectivity algorithm does not leave behind isolated nodes.

B. Evaluation of compressed sensing

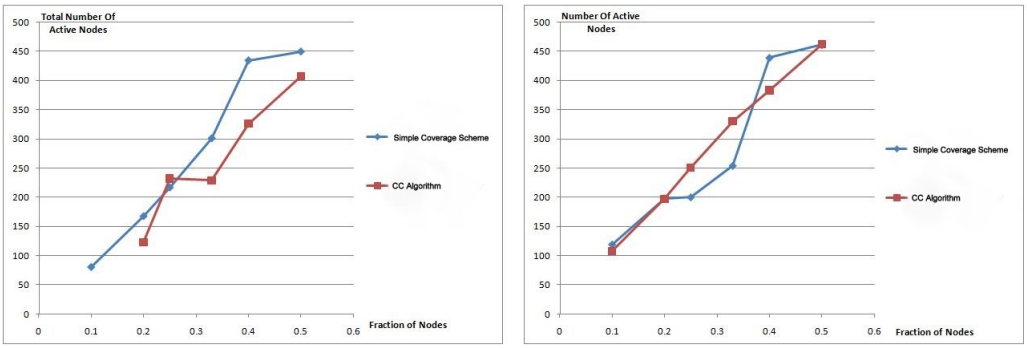
Once connectivity and coverage issues are factored in the network, we conduct experiments to test the effectiveness of compressed sensing for reducing energy consumption in sensor networks. To apply compressed sensing, we test our data acquisition scheme on synthetic as well as real data traces. We first describe how we generate the test data and then apply compressed sensing techniques to interpolate missing data. We present detailed graphs to show how applications would decide on their compressed sensing parameters.

1) Data generation: We use data generation techniques used to obtain synthetic traces. There has been a lot of research aimed at generating accurate sensor network traces since having an actual deployment is not always feasible. We choose one such tool developed by Jindal et al. [18] to build our datasets. The method for generating

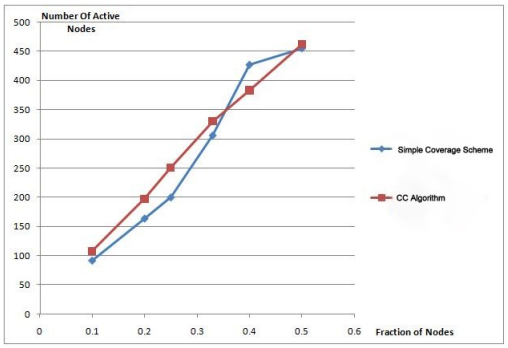


(a) Before Connecting (b) After Connection

Fig. 2. The first figure shows the sensors that were turned on in a particular instant. Blue dots represent inactive sensors and reds indicate active sensors. The first figure represents the first phase of coverage where random nodes are selected for wakeup. The second figure is similar to the first except that the connectivity algorithm has been applied on it, hence a few extra sensors can be found awake. The sensors marked with grey circles indicate some of the sensors that were awaken by the connectivity algorithm.



(a) Transmission Range=2 (b) Transmission Range=4



(c) Transmission Range=8

Fig. 3. The three figures compare the performance of the *simple coverage scheme* and the *decentralized activity planning* algorithms. In all the figures, plots of total number of awake nodes are made against the fraction of nodes which are awake for different values of γ_{avg}. All nodes were placed on a 256 by 256 unit length grid with a total of 829 nodes placed randomly. To test the system for various different kinds of deployment, plots have been made for different transmission range for the nodes which results in a different average number of neighbours. The first figure has been plotted for average number of neighbours = 9 with transmission range = 2 units. The second and the third figure have been plotted for transmission range = 4 and 8 units respectively.

these synthetic traces have been validated against sensor traces from actual deployments such as the Intel lab data trace [17] and the S-Pol radar dataset [26].The data generation tool follows a mathematical model to capture the spatial correlation in sensor network data. The tool can be used to generate a wide range of synthetic traces showing different levels of correlation. We use different levels of correlation between the network data to simulate a wide range of possible deployments. Correlation among data translates to different physical phenomenon being monitored. As it seems evident, a major advantage of using a synthetic data generator is that it can mimic several types of sensor network applications. Actual data only covers one example of data that it monitors. Apart from this, a synthetic generator can create a large or a small network in terms of number of nodes which can test out the compressed

sensing algorithm thoroughly.

2) Results: We used the synthetic data generator from the previous discussion to generate the sensor data and used compressed sensing to retrieve back missing values from the sensor data. In our experiments, we tried to change the correlation between data in each of the synthetic datasets and also tried changing the value of P during reconstruction to test the accuracy of reconstruction at various values of P. For each of the different values, we plotted the normalized mean square error versus the fraction of sensor values being used for reconstruction.

For highly correlated data, the NMSE is similar for different values of P if we have 20 percent of the sensors sending back data to the base station. Beyond this point, P=1 produces increased error compared to lower values of P (Fig. 5). For medium and low correlations, we can see that as the correlation of the data decreases, the NMSE increases. For most of the cases, having the value of P below 1 shows lower reconstruction error. Beyond the point where 80 percent of the sensors are missing, P=1 produces much higher reconstruction error compared to lower values of P.

We demonstrate the change in error with varying correlation values (Fig. 6 and 7). It is evident that with higher correlation, the NMSE is lower. Also lower values of P give better results compared to P=1.

From the results presented in Fig. 8, we can see that though the correlation among data points is not too high yet with 20 percent of the nodes sampling, we get a near-exact reconstruction of the original signal.

Regardless of the values of P and the correlation factors, one fact is evident from the results. For most sensor network deployments, we can take readings from a small fraction of nodes and reconstruct the missing node values without sacrificing accuracy. The fraction of nodes that is required depends on the error tolerance for the application. However, as we can see from our results, if 20% of the nodes in a deployment send back data to the base station, we can reconstruct the readings from all other nodes with very small reconstruction error, given we have a sufficiently large dataset. With this mechanism, energy consumption in nodes can drop to one-fifth of traditional data acquisition methods in sensor networks. We can expect the life time of each node to be 5 times longer than usual which is enormous compared to current standards.A question that remains unanswered is the overhead at the base station for running our reconstruction algorithm. Is it possible to run frequent reconstructions every time the sensors send back data? Overhead at the base terminal is quite critical as it determines the frequency with which the sensors can operate. If the time taken to reconstruct the missing values is really large, it impacts the frequency at which readings can be taken. Since the time for reconstructing the data at the base station is proportional to the size of the network, a large network will require larger reconstruction time than a smaller network. Needless to say, reconstruction time will also depend on the processing hardware. The application developer must decide the sampling frequency of an application while keeping these details in mind.

3) Actual data: Testing with synthetic data provides us with very accurate results even when the percentage of active sensors is low. To test the system on actual data, we ran compressed sensing on on actual sensor data. This gave us insight to expected results when compressed sensing is applied on real applications. We tested our algorithms on wifi data collected at the Stevens Campus and published in [23]. Since the actual values of the sensors were not available, we decided to scale down the image to a 32 $\times$ 32 grid and ran compressed sensing reconstruction on it. Fig. 9 shows the actual scaled and the reconstructed versions of the wifi data. The reconstruction is done with 10 percent active sensors and $P = 0.8$. Though the mean squared reconstruction error is 14 percent, as the number of sensors is increased to 20 percent, the error drops to 7 percent.

It is difficult to compare our improvement in efficiency with other wireless systems. In several other wireless applications, efficiency is measured as throughput of the network per joule of energy consumed [25]. However, such a metric is not appropriate for defining efficiency of our compressed sensing algorithm. In our case, throughput is measured through the accuracy of the reconstructed data while energy cost is measured through the fraction of nodes kept awake. To gain higher throughput or accuracy of reconstruction, a larger fraction of nodes need to stay awake.

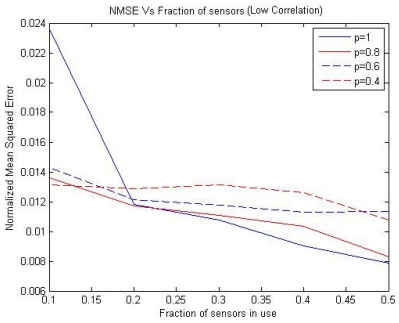
VII. Limitations of compressed sensing

We have managed to demonstrate a data management scheme that can lead to huge savings in many sensor network deployments. For WSNs that monitor some natural phenomenon with smoothly varying distributions, compressed sensing can produce multi fold increase in life time of individual nodes. Though most sensing applications fall under this category, a few may not. For example, an intrusion detection system which needs to be highly reliable may not be the best application where compressed sensing can be applied. Also, systems that generate extremely random signals cannot apply compressed sensing due to low correlation in the generated signal. Localization applications which try to track objects [27] do not fit well into the compressed sensing model as it needs to act instantaneously and every data sample is of utmost importance.

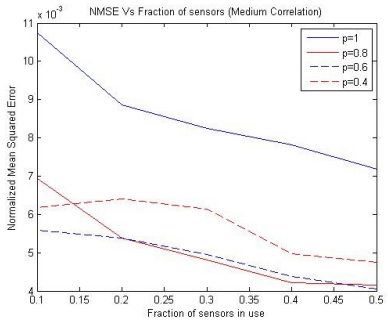
Further, the compressed sensing system needs a sufficiently large network to work. For example a grid having 20 nodes may not be the best network to test compressed sensing. Since we mainly focus on natural phenomenon, we assume that the network would be sufficiently large to allow our algorithm to select a low percentage of nodes and reconstruct without significant error.

VIII. Conclusion

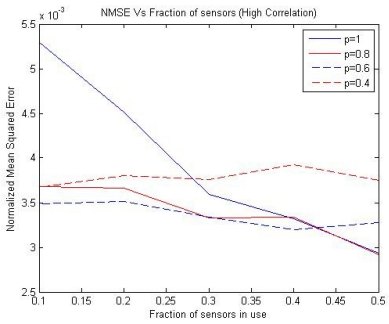
We have presented a framework that can be used to elongate the life of a sensor network deployment. We have proved that our CC Algorithm is near-optimal and successfully maintains coverage and connectivity in a sensor network. Combining our CC Algorithm with compressed sensing, we promise significant energy savings along with robustness over a well connected selection of sensor nodes. Experimentally, using simulation and test data, we have demonstrated the effectiveness of our scheme. We have also provided hints at how to employ our system on actual sensor software. The next big step for us would be to test our scheme on an actual system. A large deployment with a significant number of nodes which monitors some physical



(a) Low Correlation



(b) Medium Correlation



(c) High Correlation

Fig. 5. Normalized mean square error versus fraction of sensors transmitting data for various correlation values.

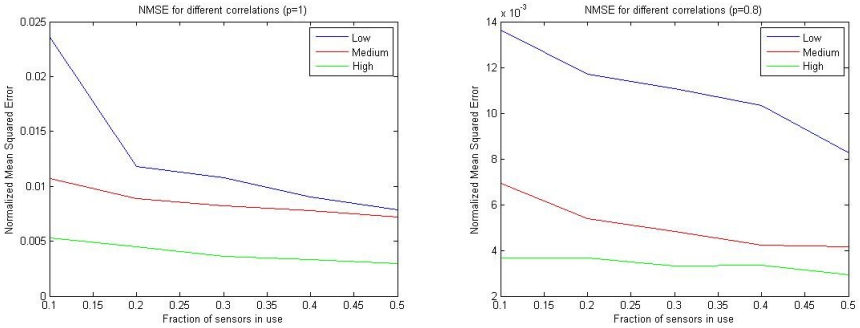


Fig. 6. Normalized mean square error versus fraction of sensors transmitting data for various value of P (P=1,0.8).

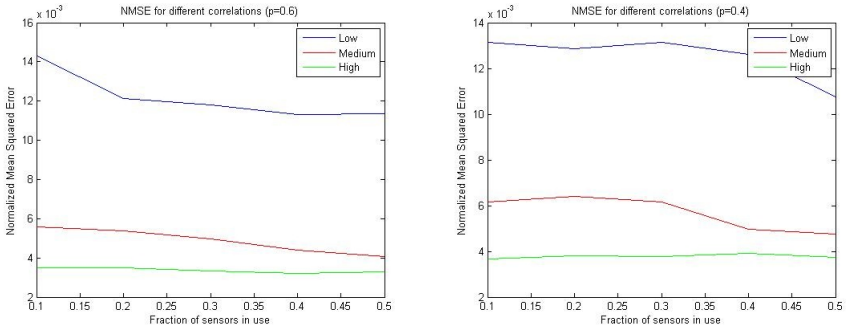
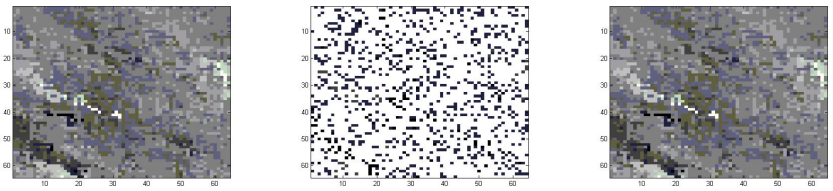


Fig. 7. Normalized mean square error versus fraction of sensors transmitting data for various value of P (P=0.8,0.6).

phenomenon having significant correlation would be the ideal platform to demonstrate our scheme.

Two important aspects of our scheme are simplicity and effectiveness. We have made our scheme as simple as possible such that it can be easily adopted. From the point of effectiveness, we point out the amount of energy savings



(a) Original Data (b) Sampled Data (c) Recovered Data

Fig. 8. The first figure shows the original data. The second image is the sampled data with 20 percent of the sensors sending back data to the base station. The third figure shows the recovered data using our reconstruction algorithm.

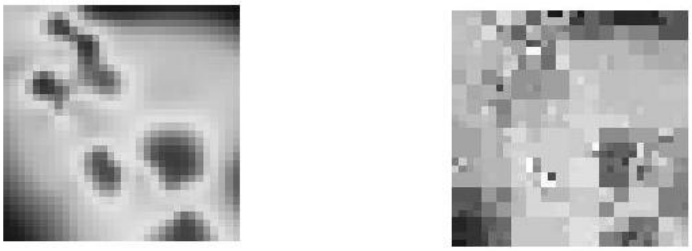


Fig. 9. Scaled and reconstructed wifi signals from the Stevens Campus. Only 10 percent of the nodes' data is used for reconstruction.

leading to increased lifetime of sensor nodes that can be extracted using a combination of our CC Algorithm and compressed sensing. Such energy savings are not possible using other micro optimization techniques.

REFERENCES

[1] Waheed Bajwa, Jarvis Haupt, Akbar Sayeed, and Robert Nowak. Compressive wireless sensing. In *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, pages 134–142, New York, NY, USA, 2006. ACM.

[2] E.J. Candes and M.B. Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21 –30, mar. 2008.

[3] Qing Cao, Tarek Abdelzaher, John Stankovic, and Tian He. The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 233–244, Washington, DC, USA, 2008. IEEE Computer Society.

[4] Qing Cao, Debessay Fesehaye, Nam Pham, Yusuf Sarwar, and Tarek Abdelzaher. Virtual battery: An energy reserve abstraction for embedded sensor networks. *Real-Time Systems Symposium, IEEE International*, 0:123–133, 2008.

[5] M. Cardei, M.T. Thai, Yingshu Li, and Weili Wu. Energy-efficient target coverage in wireless sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1976 – 1984 vol. 3, 2005.

[6] A. Cerpa and D. Estrin. Ascent: adaptive self-configuring sensor networks topologies. *Mobile Computing, IEEE Transactions on*, 3(3):272 – 285, 2004.

[7] Geoffrey Werner Challen, Jason Waterman, and Matt Welsh. Idea: integrated distributed energy awareness for wireless sensor networks. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 35–48, New York, NY, USA, 2010. ACM.

[8] R. Chartrand and Wotao Yin. Iteratively reweighted algorithms for compressive sensing. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, 31 2008.

[9] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8:481–494, September 2002.

[10] Lei Chen, Christopher Olston, and Raghu Ramakrishnan. Parallel evaluation of composite aggregate queries. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 218–227, Washington, DC, USA, 2008. IEEE Computer Society.

[11] D.L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289 –1306, apr. 2006.

[12] M.R. Duarte, M.B. Wakin, D. Baron, and R.G. Baraniuk. Universal distributed sensing via random projections. pages 177 –185, 2006.

[13] M. Elad, J.-L. Starck, P. Querre, and D.L. Donoho. Simultaneous cartoon and texture image inpainting using morphological component analysis (mca). *Applied and Computational Harmonic Analysis*, 19(3):340 – 358, 2005. Computational Harmonic Analysis - Part 1.

[14] J. Haupt, W.U. Bajwa, M. Rabbat, and R. Nowak. Compressed sensing for networked data. *Signal Processing Magazine, IEEE*, 25(2):92 –101, mar. 2008.

[15] Tian He, Chengdu Huang, Brian M. Blum, John A. Stankovic, and Tarek Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, MobiCom '03, pages 81–95, New York, NY, USA, 2003. ACM.

[16] J.L. Hill and D.E. Culler. Mica: a wireless platform for deeply embedded networks. *Micro, IEEE*, 22(6):12 – 24, 2002.

[17] INTEL. Intel lab data.

[18] Apoorva Jindal and Konstantinos Psounis. Modeling spatially correlated data in sensor networks. *ACM Trans. Sen. Netw.*, 2(4):466–499, 2006.

[19] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating Syststems Review*, 36(SI):131–146, 2002.

[20] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 2003. ACM.

[21] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[22] R. Masiero, G. Quer, M. Rossi, and M. Zorzi. A bayesian analysis of compressive sensing data recovery in wireless sensor networks. pages 1 –6, oct. 2009.

[23] G. Quer, R. Masiero, D. Munaretto, M. Rossi, J. Widmer, and M. Zorzi. On the interplay between routing and signal representation for

compressive sensing in wireless sensor networks. pages 206 –215, feb. 2009.

[24] Michael Rabbat, Jarvis Haupt, Aarti Singh, and Robert Nowak. Decentralized compression and predistribution via randomized gossiping. In *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, pages 51–59, New York, NY, USA, 2006. ACM.

[25] V. Rodoplu and T.H. Meng. Bits-per-joule capacity of energy-limited wireless networks. *Wireless Communications, IEEE Transactions on*, 6(3):857 –865, 2007.

[26] S-POL. S-pol radar data trace.

[27] Chatschik Bisdikian Lance M. Kaplan Sadaf Zahedi, Mani B. Srivastava. Quality tradeoffs in object tracking with duty-cycled sensor networks. In *Proceedings of the 2010 Real-Time Systems Symposium*, pages 160–169, Los Almos, CA, USA, 2010. IEEE Computer Society.

[28] C. Schurgers and M.B. Srivastava. Energy efficient routing in wireless sensor networks. In *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, 2001.

[29] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. In *Communications, 2001. ICC 2001. IEEE International Conference on*, 2001.

[30] K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie. Protocols for self-organization of a wireless sensor network. *Personal Communications, IEEE*, 7(5):16 –27, October 2000.

[31] Jean-Luc Starck, E.J. Candes, and D.L. Donoho. The curvelet transform for image denoising. *Image Processing, IEEE Transactions on*, 11(6):670 –684, jun. 2002.

[32] Srikanth Sundaresan, Israel Koren, Zahava Koren, and C. Mani Krishna. Event driven adaptive dut cycling in sensor networks. *International Journal on Sensor Networks*, 6:89–100, October 2009.

[33] Di Tian and Nicolas D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 32–41, New York, NY, USA, 2002. ACM.

[34] Megan Wachs, Jung Il Choi, Jung Woo Lee, Kannan Srinivasan, Zhe Chen, Mayank Jain, and Philip Levis. Visibility: a new metric for protocol design. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 73–86, New York, NY, USA, 2007. ACM.

[35] Jason Waterman, Geoffrey Werner Challen, and Matt Welsh. Peloton: coordinated resource management for sensor networks. In *HotOS'09: Proceedings of the 12th conference on Hot topics in operating systems*, pages 9–9, Berkeley, CA, USA, 2009. USENIX Association.

[36] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 70–84, New York, NY, USA, 2001. ACM.

Review and Challenges of Assumptions in Software Development

Md Abdullah Al Mamun and Jörgen Hansson
Software Engineering Division
Department of Computer Science & Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
{abdullah.mamun, jorgen.hansson}@chalmers.se

Abstract—**The problems of implicit and invalid assumptions have been identified as one of the key reasons to project and software failures. Assumptions are available in almost all aspects of the software development from human factors to different software development activities. They also have influence on software quality attributes. The aim of this article is to provide a review of existing work in assumption management and find out the assumptions related challenges that should be mitigated in order to build better systems. The results show that assumptions are concerned with many different areas of software engineering and that existing approaches suffer from the lack of scope of assumptions categories and some concerns that are impacted by the assumptions. We believe a holistic assumption management approach can mitigate assumptions related challenges by integrating concerned areas and contribute to build systems with smooth software integration and evolution.**

Keywords- assumptions; assumption management; software evolution; software and system integration; cyber-physical system

I. INTRODUCTION

Today's Cyber-Physical Systems (CPSs) intrinsically combine many domains and areas of expertise in order to achieve system goals and maximize the benefit. This demands significant interactions among people, environment, software, and hardware artifacts, which in turn dramatically increases the complexity of the system. The maximum number of concerns that the human brain can consciously process at the same time is limited. It is thus challenging for the software and system developers to consider all significant assumptions and constraints among various components to make good decisions.

An assumption is a statement that is assumed to be true and it is invalid when the assumed statement is actually not true. Assumptions are implicit when they are not documented. Assumptions can be implicit in at least two ways. First, when people are aware of the assumptions but do not document them because of lack of consciousness about the pitfalls of implicit assumptions or due to political reasons within the organization. Second, there is no awareness of the (implicit) assumptions among the stakeholders/people. Implicit assumptions thus represent tacit knowledge, which has been identified by the knowledge engineering discipline [10] as volatile and challenging to preserve and transfer.

When implicit assumptions are not documented, they get lost over time. This might happen due to that the architects forget about the assumptions they made in the past, or that the architects are not available at present. The gradual loss of architectural knowledge is a problem in the area of software architecture and this problem is known as architectural erosion or architectural drift. This scenario is also applicable for requirements, coding, and testing.

Invalid assumptions are reported as the root cause of system and project failure [17, 40, 8]. In practice, people do not make invalid assumptions intentionally or because of lack of knowledge. Today's systems often work in a complex dynamic environment with the presence of reusable components. Along with many benefits, reusable components also bring certain challenges. As COTS are developed to work in different environments, they generally do not offer a perfect match with a specific use. The use of a system as a part of a larger system is common as it is not feasible to build everything from scratch. However, a system is not always built with the intention of it being used as a part of a larger system. Moreover reusable components, COTS, middleware might have their own sets of assumptions that are implicit or not visible to the people using them. Hence, people can make assumptions about different components that are conflicting, or mismatched, with existing assumptions. The development of complex systems deals with various domains where every domain has its own practices. However, in reality an organization cannot always adopt necessary software practices of the particular domains they are engaged with. Thus, an organization might apply their existing software practices onto a new domain.

The review shows that there is a broad landscape of assumptions and several challenges have been recognized in certain areas. Initial work has been conducted to address some challenges, but we observe that there is a lack of integrated approaches toward systematic assumption management. Successful mitigation of these challenges would indeed support virtual integration of components, continuous deployment, and more loosely coupled CPSs development.

The organization of this report is as follows. Section II defines assumptions and some assumptions types used in this article, and explains how requirements, constraints, assumptions, and design rationales are connected to each other in an interchangeable way. Section III shows the literature

review of assumptions in software engineering. The challenges of assumptions are presented in section IV followed by the summary.

II. BACKGROUND

A. Definition of Assumption

The WordNet[1] lexical database defines the term assumption as follows:

- "A statement that is assumed to be true and from which a conclusion can be drawn". Example: "on the assumption that he has been injured we can infer that he will not play".
- "A hypothesis that is taken for granted". Example: "any society is built upon certain assumptions".
- "The act of assuming or taking for granted". Example: "your assumption that I would agree was unwarranted".

Now we define some assumptions types used in this paper.

Invalid vs. Valid assumptions: An assumption is considered invalid if a stated assumption is false or incorrect; it is valid otherwise, i.e., the stated assumption holds. The validity/invalidity of an assumption can most often be determined by verifying its fact without necessarily looking at any other assumptions. On the other hand, *conflicting* and *mismatched* assumptions are determined from the conjugation of more than one assumption.

Conflicting assumptions: An assumption is conflicting, if it contradicts or conflicts with one or more other assumptions. It can be both invalid and valid.

Mismatched assumptions: An assumption *X* is mismatched, if we cannot determine whether the associated components/artifacts of *X* would fulfill the fact of *X*. In other words, there is no evidence provided by the components/artifacts that could be matched against what is assumed. It can be both invalid and valid.

B. Requirements, Constraints, Assumptions, Design Rationale

It is difficult to distinctly divide requirements, constraints, assumptions, and design rationales as they often overlap each other. Sometimes these terms are used interchangeably [27] and sometimes broadly [19] to extend the coverage of their definition. In general, requirements are the expectations of the customers about a system. Constraints are facts that impose restrictions, limitations, regulations on a system. In a classical sense, requirements and constraints are sets that both the software development organization and the customers agree upon. Requirements and constraints can be both functional and nonfunctional and it is a common organizational practice to document them.

Assumptions are statements that are assumed to be true. Assumptions build the underlying reasons behind the decisions where a decision can be an architectural decision etc [35]. When decisions along with their underlying assumptions are implemented, assumptions act like constraints by restricting, limiting, and regulating the system. From this point of view, assumptions and constraints are similar. However, assumptions can be invalid from the very beginning of their existence, which is not applicable for the constraints. On the contrary, both assumptions and constraints can be invalid at any time in the future when the system evolves.

Design rationales are the motivations of design decisions where a collection of design decision explains why an architecture is in a certain form. Kruchten et al. [35] do not distinguish between assumptions and design rationale since it is difficult to make a clear distinction and consider assumptions as general denominator for the forces driving architectural design decisions. Both assumptions and rationales can be considered as elements of design decision [43]. In contrast, assumptions and constraints can also be considered as elements to capture design rationales [21].

III. ASSUMPTIONS IN SOFTWARE ENGINEERING

Figure 1 shows how assumptions can be scattered in different software development phases and still connected. A single assumption can be associated with artifacts at different levels. The scenario becomes much more complicated when assumptions are connected with requirements, design decisions, design rationales and other possible knowledge categories that are influenced by the assumptions. This review also finds assumptions in different areas of software engineering. Among them, software architecture is the area where assumptions are mostly used. Some work is directed toward implementations. Work has also been conducted in the security domain, especially at the requirements engineering level, as well as in the architectural knowledge management areas; here most often assumptions are treated informally in the form of free text.

This section first presents different *assumptions modeling approaches* then *architectural mismatch* problems due to the assumptions, followed by assumptions in the area of *requirements engineering* and *software security*. The end of this section focuses on assumptions in the *knowledge management* discipline and *software development processes*.

A. Assumption Modeling

There have been attempts on modeling assumptions. We have divided them into two classes that are formal and semi-formal. By formal, we mean those modeling approaches that formalize the statement or the fact of an assumption along with other attributes. Approaches that describe the fact/statement of the assumptions as free text but other attributes like assumption category description, source, impact, criticality, tractability information etc are structured are termed as semi-formal.

[1] Website: http://wordnetweb.princeton.edu/perl/webwn

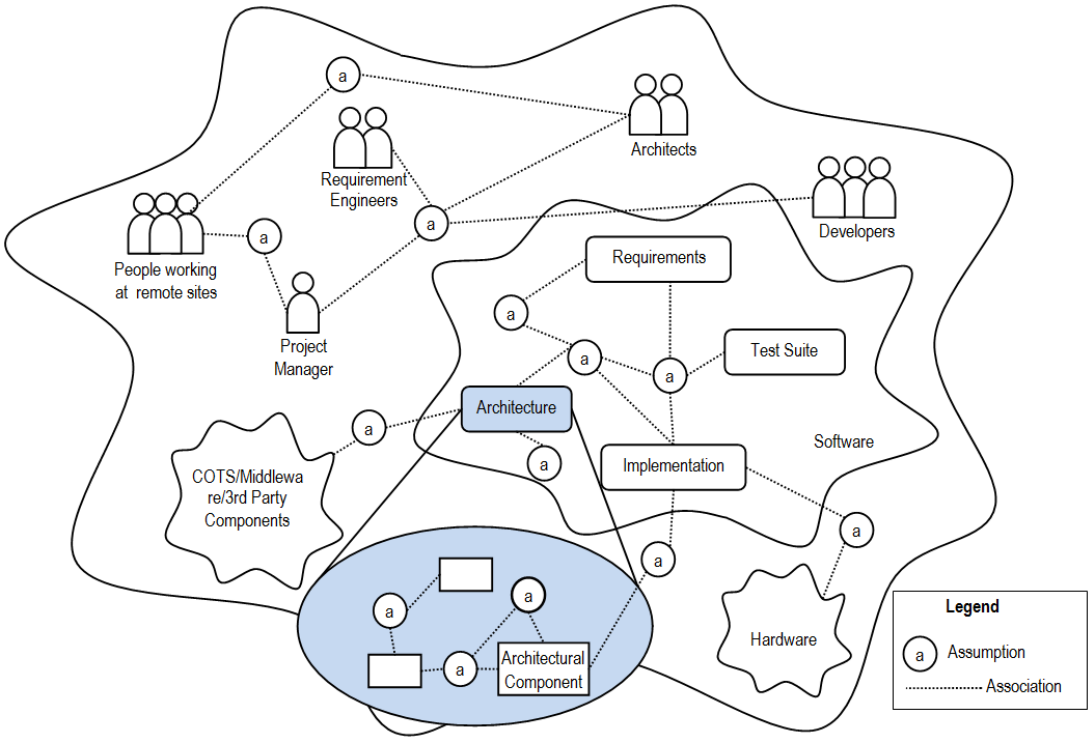


Figure 1. Assumptions in Software and System Development

The idea of informal assumptions modeling is not so appropriate. However, we can say assumptions are informal when they are documented without proper structure, completely in free text in the form of comments. The advantage of formally modeled assumptions is that they are potentially machine-checkable. However, formal approaches have suffered from limited scope, as it is challenging to formally model assumptions of concerning, e.g., managerial and environmental aspects.

1) Formal Approaches

A framework toward assumption management has been developed by Tirumala [2]. They have developed a language for documenting assumptions in a machine-checkable format where the assumptions and the guarantees for the assumptions are encoded as part of the architectural components. The framework is capable of dealing with both the architecture and the implementation. In addition to the static assumptions, the framework also supports dynamic validation of assumptions. Even though, automated checking of invalid assumptions is a big advantage for the large-scale, complex system development but the scope of the targeted assumptions is limited to the technical category. The framework is implemented for the Architecture Analysis and Design Language (AADL) [31], which is an Architecture Description Language (ADL)

2) Semi-Formal Approaches

Lewis et al. [14] have developed a simple assumption management system prototype for recording and extracting assumptions from code written in Java into a repository. The assumptions are written in the code using XML and saved into the repository using an assumption extractor. This web-based assumption management system offers browsing and searching of assumptions with given criteria. The stored assumptions are then reviewed by a person who acts as a validator. The management system also maintains system and project related information like users, roles, projects and types of assumptions. The scope of this prototype is limited to assumption management at the implementation level.

A meta-model for explicating assumptions in the software architecture has been developed by Lago and Vliet [33]. The model is able to handle assumption dependencies between the product feature model and the architectural model. They have worked with a software product family architecture implementing variability to achieve flexibility. They introduce the term invariability and argue that invariability should also be modeled along with variability to let the model express what cannot be changed. As assumptions are somewhat related to the constraints that impose limitations on the system behavior, assumptions would tell us what we cannot change or what would be challenging to change. Thus it is possible to achieve architectural invariability through explicit assumptions modeling.

Ordibehesht [16] has implemented an assumptions modeling method for explicating assumptions in the AADL. The modeling method consists of an assumption specification meta-model for structuring assumptions information and an assumptions specification approach to specify the meta-model together with the architecture descriptions. The meta-model contains dependency information between the assumptions and the architectural components in order to facilitate traceability.

B. Architectural Mismatch

Garlan et al. [8] report that implicit assumptions are the root cause of widespread software reuse problems. They used the term "architectural mismatch" to express the idea that reusable architectural components make implicit assumptions on other parts of the architecture without being validated. As these assumptions are implicit, they often conflict others thus making the system unusable. Since the assumptions are usually implicit, it is difficult to analyze them before the system is built. They identify four categories of assumptions that can contribute to architectural mismatch in terms of components and connectors. They are:

- Nature of the components (control model, data model)
- Nature of the connectors (protocol, data model)
- Global architecture structure
- Construction process (development environment and built)

Even though the authors identify implicit assumptions as the root cause of the architectural mismatch and suggest to make assumptions explicit, they believe explicating assumptions alone cannot completely fight architectural mismatch, and thus they suggest additional solutions such as use of orthogonal sub-components, create bridging techniques like mediators, and develop source of design guidance.

Garlan et al. [9] have revisited the challenges of architectural mismatch with the advancement of time in comparison to their previous study. They discuss three basic techniques with architectural mismatch namely mismatch prevention, mismatch detection and mismatch repairing along with the approaches to solve mismatch problems. Trust, dynamism, architecture evolution, and architecture lock-in are reported as new challenges in this field.

Cai et al. [25] have proposed an approach to identify architectural mismatches resulting from invalid assumptions in an event-based system. The semi-automatic approach is implemented with a customized version of the model checker Bandera/Bogor tool pipeline applied on a Java version of SIENA event service.

Architectural mismatch has been analyzed considering assumptions from a different perspective by Uchitel and Yankelevich [42]. Assumptions are viewed as connections that components make about each other. Here the term connection means something beyond the classical sense of connections. This view originated from Parnas [11] who extended the view of the connections from control or information transfer points to components' associations. Uchitel and Yankelevich [42] have performed behavior analysis to detect architectural mismatches where a labeled transition system (LTS) is used to model process behavior. They also discuss some issues of extending ADLs to include assumptions.

Lemos et al. [36] have focused on architectural mismatch tolerance, i.e., an approach that would help the system to tolerate architectural mismatches during run-time rather than dealing with the mismatches during the development time. They apply the general principle of fault tolerance to deal with architectural mismatches.

C. Assumptions and Requirements

It is generally considered that a large number of assumptions lie around the requirements to weave a complete picture of the system. Lamsweerde [4] describes this scenario as 'assumptions underlie the requirements iceberg'. Thus assumption management is often considered to be closely intertwined with the field of requirements engineering.

Fickas and Feather [41] argue that invalid environmental assumptions cause the system to evolve. Therefore, it is interesting to know when such assumptions get invalid while the system is executing. The concept of this approach is to monitor the underlying assumptions of the requirements to realize whether the requirements are met by the system while it is executing. When certain assumptions are found invalid, their corresponding/dependant requirements are considered to be compromised.

A temporal mathematical model has been proposed by Miranskyy et al. [1] to describe the relationship between requirements and assumptions in the context of risk prediction associated with assumptions failure. In the proposed model, the relationship between requirements and assumptions are captured using a Boolean network. A stochastic process is used to model the validity of the system over time.

D. Software Security

Assumptions in the security domain are known as trust assumptions because trust and trustworthiness build the foundations of security [20]. Trust assumptions might have significant impact on the system's security. An example of an implicit or explicit trust assumption can be *compilers are not vulnerable to systems security*. However, this assumption can be invalid as Thompson [23] shows how the compilers can introduce trapdoors to compromise the security of a system. Thus it needs to be reviewed for its validity. Viega et al. [22] discuss the potential risks of trust assumptions towards the software security and the origins of invalid trust assumptions like user input, client application, execution environment, software developers, users etc. They suggest adopting a general assumption management strategy, i.e., assumption identification, documentation, and analysis in order to minimize the security risks due to invalid assumptions.

Haley et al. [5] discuss trust assumptions from the view of the requirements engineers in the context of security. The requirements engineers make assumptions when analyzing the security requirements. The scope of the analysis, security requirements' derivation, and in some cases how functionality is realized are affected by the trust assumptions. A representation of trust assumptions is showed by Haley et al. [5] along with a case study to examine the impact of trust assumptions on software using secure electronic transaction specification. In more recent work, Haley et al. [6] attempted to answer to the question "how to determine adequate security requirements for a system". In this work, they considered assumptions as one of the three criteria that should be satisfied to determine adequate security requirements. A lightweight approach for mitigating security risks based on trust assumptions is proposed by Page et al. [44] that can be used within agile development environments. This work is directed

toward detection and mitigation of security risks. They developed a model where the concept of trust assumptions is used to derive obstacles, and the concept of misuse cases is used to model the obstacles.

E. Architectural Design Decision & Rationale Management

The literature of architectural knowledge identifies four primary views on architectural knowledge namely pattern, dynamics, requirements and decision-centric view [37]. The decision-centric view emerged as the importance of preserving architectural design decisions and rationales behind the decisions were realized [24, 20], and there seems to be a gradual shift of viewing software architecture as the high-level structure of components and connectors (i.e. the end result) to the rationale behind the end result [37].

Since an architecture is built based on certain design decisions, it is also seen as a collection of design decisions. Going a level further down, every design decision is made based on some rationales. Thus design decisions along with the rationales explain why an architecture is in a certain form. Assumptions are considered in both design decision and design rationale management. With the change of the perspective, something identified as an assumption may be seen as a design decision. It should be noted that, e.g., Kruchten et al. [35] do not distinguish between assumptions and design rationale as they have found it difficult to make a clear distinction. Rather, assumptions are seen as general denominator for the forces driving architectural design decisions [35].

Both assumptions and rationales are considered as elements of design decision by Dingsøyr and Vliet [44]. They describe assumption as the underlying facts about the environment in which the design decision is taken and rationale as the explanation of why the specific decision was taken. A pragmatic approach to capture design rationale has been proposed by Tyree and Akerman [21]. They include assumptions and constraints along with other elements in a template developed to capture design rationale. A rationale-based architecture model has been developed by Tang [3], which represents design rationale, design objects and their relationships. The model is able to capture both qualitative and quantitative design rationale where assumptions and constraints are considered as the drivers of design rationale. Even though assumptions are reported as one of the key factors driving design decisions and rationales, the discussed literature captures assumptions as text in natural language without further structure.

F. Other Work

Ostacchini and Wermelinger [18] have experimented with a lightweight assumption management method on agile development over three months with the result that assumption management can be integrated with the agile developments. They have recorded 50 assumptions where more than 50% are organizational or managerial. They suggest engaging the management/managers into the assumption management process as over 15% of the managerial assumptions were identified as invalid during the three months observation period.

Roeller et al. [38] have worked on the recovery of assumptions from a system that was built in the past without the assumptions being documented. At first, they reviewed financial reports, documentations, development process information extracted from the version control system and source code to identify error prone modules in the studied system. Tools were used to extract different metrics from the version control data and source code. Furthermore, interviews were performed with the architects and the developers to discuss the selected modules in order to capture the implicit assumptions.

The authors express that it is challenging to recover assumptions from a system without having a thorough understanding of the system. Moreover, the unavailability of key people and stakeholders, change in responsibilities in the project, and identifying the right person knowledgeable to specific artifacts are also quite challenging to deal with. Similarly, Garlan et al. [8] have reported from their experience with AESOP that assumption recovery could be expensive thus impractical or even impossible for legacy systems when the source code is not available.

IV. Challenges of Assumptions

CPSs are multidisciplinary in nature, addressing engineering issues at software, system, and mechanical level. Furthermore, CPSs demand a lot of interaction among the concerned components and environments. They are often also highly complex and tightly coupled systems. The development of CPSs is also often distributed in nature. Thus, it is unrealistic or not feasible to co-locate the entire CPSs development process. It is thus desirable to facilitate stronger integration approaches, and weaken strong coupling and dependencies in the development as well as the architecture level. Powerful management of assumptions has a strong potential in addressing identified concerns, e.g. the concepts of *assumptions-aware components* and *separating assumptions from artifacts (section A and B)*. Other challenges identified include evidence-*based software engineering* (section C) and *assumptions in the organization's safety culture* (section D). The *holistic assumption management system* (section E) is the foundation to manage assumptions in an efficient way throughout the entire software development process. *Assumption-based trust building* (section H) concept is applicable for human factor *trust* in global software developments thus it would also support the development of CPSs.

A. Assumption-Aware Component Development

CPSs are characteristically tightly coupled, which incurs certain inflexibilities in the system development. It would be desirable to develop components more independently, e.g., loosely coupled but still composable, which could be facilitated through self-descriptive architectural components or executable software components. The assumption-aware components should be able to describe their own structure and details about what the components expects from the other components and what the components provide for the other components with a standard syntax that is understandable by all the parties. As assumptions build the leaf-level knowledge of the artifacts, it is

possible to encode the inter-component dependencies and relationships as assumptions into the components. This makes the components assumption-aware and would offer better static and dynamic composability of such components by minimizing architectural or component mismatches. Assumption-aware components would also support the concept of virtual integration [32]. When the architecture and its substructures (e.g., components) are designed as assumption-cognizant, continuous deployment would be possible with the presence of monitors that would look for assumption-based conflicts and mismatches among the components. During composition, it is obvious that architectural mismatches may occur due to conflicting or mismatched assumptions that can be mitigated to some extent with the architectural mismatch tolerance techniques.

B. Separation of Assumptions from Artifacts

With the advent of assumption-aware component development, COTS and middleware developers would prefer to supply the assumptions related to the COTS or middleware to the customers without supplying the actual architecture or code so that the COTS or middleware can be tested whether they are composable with the customers' system or not. This concept also supports virtual integration.

C. Evidence-Based Software Engineering

In software engineering, *complacency* is a challenging problem to tackle. People suffer from complacency because of the lack of evidence. A study of five major spacecraft accidents reports complacency as the root cause of the studied system failures [29]. Another extensive study reports "*lack of evidence*" as the key problems of dependable software systems [12]. This study also proposes the idea of certifiably dependable systems that means a dependable system can be certified according to the available evidences supporting the dependability claims.

Assumptions build the leaf-level knowledge and forensic evidences of any artifact. They are able to reason why an architectural component is in a certain form. They can explain why a variable is not memory protected. In fact, assumptions make the leaf-level fingerprints of the decisions that we make while developing a system. From this point of view, assumptions are underlying evidences that can be used as a metric to measure the dependability of software components or systems.

D. Assumptions in the Organization's Safety Culture

The space shuttle Challenger disaster is a well-known case of system failure due to mismatched assumptions. The investigation report [40] of this disaster shows why assumptions should be added to the organization's safety culture. Before the launch of the shuttle, the engineers warned about the mismatched assumptions, which the management repeatedly ignored. The flight was already delayed with different issues. The management was afraid to delay it further probably because project delay negatively shows the efficiency of the management. Again, twelve years after the Challenger disaster, in 1996, we observed the explosion of Ariane 5 during its maiden flight. Such cases suggest considering assumptions in the organization's safety culture. An invalid/conflicting/mismatched assumption should be considered in the decision support system according to its criticality, priority, and impact. Assumption-based hazard analysis and Failure Mode Effect Analysis (FMEA) can reduce the risks of invalid, conflicting and mismatched assumptions.

E. A Holistic Assumption Management Approach

The span of assumptions in software development is not limited to any specific phases rather it is widespread. Available assumption management frameworks either focus on a narrow scope of assumptions types being very formal by modeling the assumptions in a machine-checkable format [2] or cover a wide variety of assumptions in a semi-formal approach that is not automated [33]. Moreover, existing approaches are not capable of providing an integrated solution toward assumptions by covering different software development phases, domains, COTS, and middleware. Since it is challenging and even not feasible to formally model all types of assumptions, we believe that building a flexible assumption management framework that would facilitate documenting assumptions in both machine-checkable and human-readable format is necessary.

In addition to the assumption-based services provided to different software development phases, the assumption management framework should also provide services to other frameworks that might be benefited from the assumptions, e.g., knowledge management, security, safety, etc. The knowledge management frameworks use assumptions as the underlying motivators for the design decisions or design rationales. For example, the assumption management framework can send a warning message to a knowledge management framework indicating that some of the design decisions or design rationales are subject to review, because their underlying assumptions are identified as invalid or conflicting.

F. Prioritization of Assumptions

A commonly raised argument against documenting assumption probably is "*assumptions can be anywhere; it is not feasible to document and maintain all of them simply because they are too many*". This argument is realistic and probably true. However, given the history we know that invalid assumptions can result in catastrophic consequences. While it seems infeasible to document all assumptions given time and budgetary constraints, we need to develop methods to prioritize assumptions in order to maximize the benefit over the cost.

Prioritization methods can help to identify/document important assumptions. They can also help selecting important assumptions among the identified assumptions that would be maintained throughout the software life cycle. Assumptions can be prioritized according to the domains, project types, technology used to build software system, software process, criticality, etc.

G. Assumption-based Verification and Validation

Assumptions are reported as one of the key problems failing the systems. It is expected that managing assumptions would reveal many defects earlier. Assumptions that are formally documented in the source code can be automatically

checked both statically and dynamically. However, there is no guarantee that a manual checking of assumptions in the implementation or automated/manual checking of architectural assumptions would prevent all defects related to these checked assumptions. From the testers' point of view, a good place to sniff the system for possible defects is where the defects are. However, it is not easy to know beforehand where the defects actually are. When finding the defects, the earlier in the system development life cycle is generally better. Therefore, it is quite reasonable to develop test cases motivated by the assumptions and then test the system with them. If we can automate these tests, they can be applied repeatedly in a cost effective way as the system evolves. Moreover, the architectural assumptions can be used to develop test cases, scenarios to review the architecture. The same concept is applicable for the verification of requirements.

H. Assumption-Based Trust Building and Maintenance

Software development is human-centric which involves a dimension of complexities toward successful management of software projects. The increasing popularity of distributed software development further boosts these complexities. In a globally distributed project people from different geographical location, society, culture, organizations and time zones take part in developing software.

Trust has been identified as one of the key success factors of distributed software projects [26, 28]. Face-to-face meeting and socialization are primary trust building activities that are easily achievable for the co-located team members. Time and budgetary constraints often do not allow face-to-face meeting among the distributed teams [30]. In distributed software projects, temporal, geographical and socio-cultural disparities obstruct communication, coordination and cooperation among the remote team members [34], which in turn contribute to developing mistrust among them. Figure 1 shows assumptions between people working in a software project. Whenever the assumptions do not match the reality, we become unsatisfied and conflict arises. Moe and Šmite [28] report the key factors that cause lack of trust.

As a solution to the problems, researchers suggest to take necessary action to mitigate the factors contributing to lack of trust [13, 7, 15, 26]. Others suggested considering a flexible and adaptable software development method that facilitates more communication and coordination among the team members [39].

Thus it can be argued that invalid assumptions may be a source of mistrust. Therefore, the solution should be directed toward where the problem originates. The idea of assumption management of human factors can also be applied, in which case it can build and maintain trust among the development teams. Thus it would be possible to reduce the key factors in a cost effective way that act as hindrances toward building and maintaining trust in the distributed working environment.

V. SUMMARY

Making assumptions is unavoidable when developing software systems. This article has provided an overview showing that assumptions are used in a number of different areas of software and system engineering. It is clear that there is a lack of integrated approaches toward systematic assumption management, enabling quantitative analysis and checks of assumptions, which would ultimately mitigate the key challenges associated with the assumptions. Mitigation of the challenges would support virtual integration of components, continuous deployment and more loosely coupled CPSs development. A holistic assumption management framework can offer different services to such other frameworks such as accessing the assumptions and their properties, on-request assumptions validation, on-request assumption updates, report errors, warnings, etc.

Currently, we are working on building a meta-model to capture assumptions at different system levels, e.g., component, subsystem, and system. Initially, we focus on the software and system architecture, with a particular focus on formal architecture specifications captured in an architecture modeling language such as AADL or OMG MARTE. The goal is to capture assumptions explicitly in the architecture model and conduct automated and quantitative analysis of the model. Thus part of the scope is to generate methods and tools for assumption-based verification and validation conducive to enabling smooth integration and continuous deployment of software systems.

REFERENCES

[1] A. Miranskyy, N. Madhavji, M. Davison, and M. Reesor, "Modelling assumptions and requirements in the context of project risk," in *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, 2005, pp. 471-472.

[2] A. S. Tirumala, "An assumptions management framework for systems software," Ph.D. thesis. 2006.

[3] A. Tang, "A rationale-based model for architecture design reasoning," Ph.D. thesis. 2007.

[4] A. van Lamsweerde. Requirements engineering in the year 00: a research perspective. In ICSE '00: Proceedings of the 22nd international conference on Software engineering, pages 5–19. ACM Press, 2000.

[5] C. B. Haley, R. C. Laney, J. D. Moffett, and B. Nuseibeh, "Using trust assumptions with security requirements," *Requirements Engineering*, vol. 11, no. 2, pp. 138-51, 2006.

[6] C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh, "Security requirements engineering: A framework for representation and analysis," *IEEE Transactions on Software Engineering*, pp. 133–153, 2008.

[7] D. Bandow 2001. Time to create sound teamwork. *Journal for Quality and Participation* 24(2): 41.

[8] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: Why reuse is so hard," *Software, IEEE*, vol. 12, no. 6, pp. 17–26, 1995.

[9] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: why reuse is still so hard," *Software, IEEE*, vol. 26, no. 4, pp. 66–69, 2009.

[10] D. Hislop, Knowledge management in organizations: A critical introduction. Oxford University Press, 2005.

[11] D. Parnas. Information distribution aspects of design methodology. In *Proceedings of the 1971 IFIP Congress*, Amsterdam, November 1971. North Holland Publishing.

[12] D. Jackson, M. Thomas, and L.I. Millett, *Software for dependable systems: sufficient evidence?*, Natl Academy Pr, 2007.

[13] E. Rocco 1998. Trust Breaks Down in Electronic Contexts but Can be Repaired by Some Initial Face-to-face Contact. ACM: Los Angeles, CA, New York.

[14] G. A. Lewis, T. Mahatham, and L. Wrage, "Assumptions management in software development," 2004.

[15] G Piccoli, B. Ives 2003. Trust and the unintended effects of behavior control in virtual teams. *Mis Quarterly* 27(3): 365–395.

[16] H. Ordibehesht, "Explicating Critical Assumptions in Software Architectures Using AADL," *rapport nr.: Report/Department of Applied Information Technology 2010: 115*, 2010.

[17] Inquiry Board. ARIANE 5 – Flight 501 Failure. Inquiry Board report, http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf (March 2011)

[18] I. Ostacchini and M. Wermelinger, "Managing assumptions during agile development," in *2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK 2009), 16 May 2009*, Piscataway, NJ, USA, 2009, pp. 9-16.

[19] J. A. Dewar, Assumption-Based Planning; A Planning Tool for Very Uncertain Times. DTIC Document, 1993.

[20] J. Bosch, "Software Architecture: The Next Step," in *Software Architecture*, vol. 3047, F. Oquendo, B. C. Warboys, and R. Morrison, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 194-199.

[21] J. Tyree and A. Akerman, "Architecture decisions: demystifying architecture," *IEEE Software*, vol. 22, no. 2, pp. 19- 27, Apr. 2005.

[22] J. Viega, T. Kohno, and B. Potter, "Trust (and mistrust) in secure applications," *Communications of the ACM*, vol. 44, no. 2, pp. 31-6, Feb. 2001.

[23] K. Thompson, "Reflections on trusting trust," *Communications of the ACM*, vol. 27, no. 8, pp. 761–763, 1984.

[24] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[25] L. R. Cai, J. S. Bradbury, and J. Dingel, "Discovering Architectural Mismatch in Distributed Event-based Systems using Software Model Checking," 2006.

[26] M Ali Babar, JM Verner, PT. Nguyen. 2006. Establishing and maintaining trust in software outsourcing relationships: an empirical investigation. *Journal of Systems and Software* 80(9): 1438–1449.

[27] M. Spiegel, J. Reynolds, and D. C. Brogan, "A case study of model context for simulation composability and reusability," in *Proceedings of the 37th conference on Winter simulation*, Orlando, Florida, 2005, pp. 437–444.

[28] N. B. Moe and D. Šmite, "Understanding a lack of trust in Global Software Teams: a multiple-case study," *Software Process: Improvement and Practice*, vol. 13, no. 3, pp. 217–231, 2008.

[29] N.G. Leveson, "Role of software in spacecraft accidents," *Journal of spacecraft and Rockets*, vol. 41, 2004, pp. 564–575.

[30] P.-A. Quinones, S. R. Fussell, L. Soibelman, and B. Akinci, "Bridging the gap: discovering mental models in globally collaborative contexts," in *Proceeding of the 2009 international workshop on Intercultural collaboration*, Palo Alto, California, USA, 2009, pp. 101–110.

[31] P. Feiler, D. Cluch, J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction. CMU/SEI-2006-TN-011.

[32] P. Feiler, L. Wrage, and J. Hansson, "System Architecture Virtual Integration: A Case Study," Software Engineering Institute, CMU, Technical Report CMU/SEI-2009-TR-017, Nov. 2009.

[33] P. Lago and H. van Vliet, "Explicit assumptions enrich architectural models," in *27th International Conference on Software Engineering, 15-21 May 2005*, Piscataway, NJ, USA, 2005, pp. 206-14.

[34] P. J. Ågerfalk, B. Fitzgerald, H. Holmström, B. Lings, B. Lundell, and E. Ó. Conchúir, "A framework for considering opportunities and threats in distributed software development," in *Proceedings of International Workshop on Distributed Software Development*, France, 2005, pp. 47–61.

[35] P. Kruchten, P. Lago, and H. Vliet, "Building Up and Reasoning About Architectural Knowledge," in *Quality of Software Architectures*, vol. 4214, C. Hofmeister, I. Crnkovic, and R. Reussner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 43-58.

[36] R. De Lemos, C. Gacek, and A. Romanovsky, "Architectural mismatch tolerance," pp. 175–194, 2003.

[37] R. Farenhorst and R. C. Boer, "Knowledge Management in Software Architecture: State of the Art," in *Software Architecture Knowledge Management*, M. Ali Babar, T. Dingsøyr, P. Lago, and H. Vliet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 21-38.

[38] R. Roeller, P. Lago, and H. van Vliet, "Recovering architectural assumptions," *Journal of Systems and Software*, vol. 79, no. 4, pp. 552-573, Apr. 2006.

[39] R Sabherwal 1999. The role of trust in outsourced IS development projects. *Communications of the ACM* 42(2): 80–86.

[40] R. William P. et al., "Report of the Presidential Commission on the Space Shuttle Challenger Accident," U.S. Government Accounting Office, Washington, D.C., 1986.

[41] S. Fickas and M. S. Feather, "Requirements monitoring in dynamic environments," in *Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on*, 1995, pp. 140-147.

[42] S. Uchitel and D. Yankelevich, "Enhancing architectural mismatch detection with assumptions," in Engineering of Computer Based Systems, 2000. (ECBS 2000) Proceedings. Seventh IEEE International Conference and Workshopon the, 2000, pp. 138-146.

[43] T. Dingsøyr and H. Vliet, "Introduction to Software Architecture and Knowledge Management," in *Software Architecture Knowledge Management*, M. Ali Babar, T. Dingsøyr, P. Lago, and H. Vliet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1-17.

[44] V. Page, M. Dixon, and I. Choudhury, "Security risk mitigation for information systems," *BT technology journal*, vol. 25, no. 1, pp. 118–127, 2007.

A Criticality Decomposition Architecture to Integrate Encrypted Sensor Data in the Smart Grid

Dionisio de Niz and Lutz Wrage
dionisio@sei.cmu.edu, lwrage@sei.cmu.edu
Software Engineering Institute
Carnegie Mellon University

***Abstract*—In this paper we discuss the challenges that the integration of encryption protocols can impose on the scheduling of real-time systems in the smart grid. In order to address these challenges, we present a new task model and scheduling that takes advantage of the predictable bimodal nature of the execution of the tasks in the system. This nature allows us to use stream ciphers to decompose the computation of the encryption into two parts: a key stream pre-computation and a fast encryption computation using the key stream bits. This structure takes advantage of the fact that the trailing part of our bimodal task is not always executed (e.g., the actuation). When this happens we execute the keystream pre-computation to calculate and save values (key bits) to be used by the trailing part of the task in a future activation. We call this scheme *computation buffering*. The values produced during computation buffering reduce the execution time needed in the trailing part. Then we show how this scheme reduces both the utilization of the task (and the taskset) and the response time of the trailing part of the task. We then present the mapping of this task model to the multi-frame scheduling model and the modifications to the response time calculation. Throughout our discussion we use an example from the smart grid to illustrate both the challenges and the benefits of our solution.**

I. Introduction

The use of open communication standards (e.g., Internet protocols over corporate networks) for telemetry and control in the electric grid demands new security measures to counteract potential attacks. These measures involve the use of encryption techniques to protect the confidentiality and the integrity of information. Unfortunately, the use of encryption functions in latency-constrained control loops can enlarge the response time of this loop beyond its deadline. This is the case of control loops where a large number of Phasor Measurement Units (PMU) communicates with a central controller to evaluate the stability of the system and take corrective actions (e.g., tripping a generator) in case of instability. This is known as a Wide Area Control System (WACS) [5].

Figure 1 depicts a simplified version of the sample WACS presented in [5] that emphasizes the computation that happens inside the controller. This figure presents a number of PMUs that send encrypted messages over a WAN (e.g., the corporate WAN). The controller internally decrypts the messages and evaluates whether the system has reached an instability threshold. If that is the case, then the controller calculates the instability details (e.g., origin and potential cause) and the proper action (e.g., what generator to trip) to correct it. Then it sends the encrypted actuation (e.g. tripping) message to the appropriate generator. In order for the actuation action to be effective, it must be executed within the appropriate time latency. For instance, [4] mentions that a protective relaying action should happen within 4 ms. This means that the system needs to collect the system state data (e.g. from PMUs) and evaluate a proper action within this time. Unfortunately, while an increase use of smart measuring units (PMUs) can increase the quality of the system state evaluation (whether is stable or not), it can also increase the computation time necessary to calculate the state and action of the system. This situation gets worse whenever we need to add the security mechanisms (encryption) to prevent attacks. To evaluate whether the computation in the controller in Figure 1 completes within the appropriate deadline it is possible to use schedulability algorithms and response time analysis. In particular, for a set of N periodic tasks running in a controller processor we can determine if they can complete before its next period elapses if they are scheduled under rate-monotonic scheduling [8] and the following inequality is true:

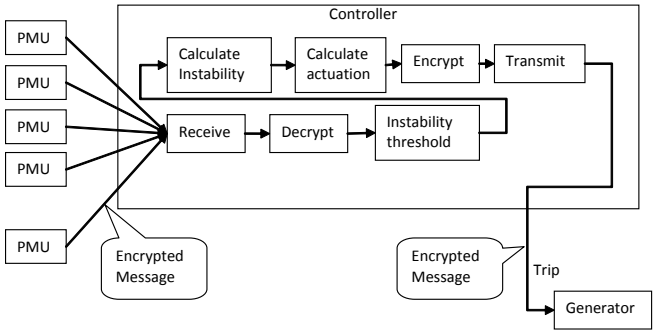


Figure 1. WACS Example

This work was supported by the U.S. Department of Energy (DoE).

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq UB$$

where:

- UB is 1 for harmonic tasksets and $N(2^{\frac{1}{N}} - 1)$ otherwise,
- C_i is the worst-case computation time of task i, and
- T_i is the period of the task.

The encryption and decryption functions can prevent the controller from satisfying its deadlines and add unacceptable latency to the control/protection actions. For instance, if 10 controller tasks (or threads) like the one shown in Figure 1 are run in a computer and each task takes 0.5 ms to run (in the worst case) with a periodicity of 10ms then we know that

$$\sum_{1}^{10} \frac{0.5}{10} = 0.5 \leq 1$$

This means that the tasks can execute at a periodicity of 10 ms and finish within 10 ms of the start of the execution. However, if the tasks need to be executed more frequently, e.g., every 4ms as required by relaying operation according to [6], then we have that

$$\sum_{1}^{10} \frac{0.5}{4} = 1.25 > 1$$

Given that the utilization is over 1, it is not possible to run these tasks at this periodicity. In this case, the encryption mechanisms contribute to the worst-case execution time and hence are part of the problem.

In this paper we present a decomposition structure and scheduling framework to reduce the penalty of computation that do not depend on the instantaneous state of the system. This is the case of encryption using stream ciphers [1] that can pre-compute keystream to be used at a later time. In addition, we use the multiframe task model to take into account that the trailing part of the task (e.g. actuation) is not always executed. We show that using this structure is possible to reduce the utilization of the taskset and improve the response time of the critical execution path.

A. Related Work

Research related to this paper can be divided in two types. On the one hand, we have extensions to the real-time schedulability model that support multiple execution times. On the other hand, we use stream ciphers to decompose encryption functions into two parts that are executed at different times depending on the required reaction time of the event at hand. Extensions to the traditional rate-monotonic model [8] to support variations in the execution time have been extensive. These variations include the multiframe task model [2] that represent the execution time as a sequence of numbers instead of a single number. This sequence represent the different execution times that the task can have that is both predictable and follows a decreasing order, i.e., starts with the largest one. This property is called *Accumulatively Monotonic*. With this model the authors proved that tasks are schedulable if their critical instance are schedulable. This work was later generalized in [3] decoupling the periods and deadlines of tasks and adding sporadic task arrivals. This model was later generalized by [4] with the *recurring real-time* (RTT) task removing the sequential restriction of the execution times. More recently [10] extended the model to support arbitrarily directed graphs that represent different possible execution parameters of the jobs of a task. In our work, we use the basic multiframe model as the basis to restructure the architecture of the software and enable computation buffering. This allows us to go beyond an extension to the timing model into a new architectural pattern that improved the overall schedulability of the system.

From the number of research efforts related to stream ciphers we found two that are close to our work. In [7] the authors present a new fast encryption scheme for real-time video called *Chaotic Video Encryption Scheme* (CVES). This scheme is a special solution for video encryption that use chaotic maps to make pseudo-random permutations of masked video. While this work is similar to ours, this is specialized for video and does not include a real-time scheduling framework to verify the schedulability of the system. In [9] the authors present the encryption technique used in MPEG video. The core of the technique is the encryption of the "I" frames. The effectiveness of the technique is based on the importance of the "I" frame, i.e., without it the video cannot be reconstructured. Contrasting to our scheme, this technique is very specific for MPEG video encryption and cannot be generalized.

The rest of the paper is organized as follows. Section II introduces the structure of our system and discusses one of our key elements: *computation buffering*. Section III discusses how we use stream ciphers and computation buffering to reduce the utilization of the system. Section IV presents the new schedulability test for our new scheme. Finally Section V present our conclusions.

II. Predictable Bimodal Execution

The system depicted in Figure 1 exhibits a predictable bimodal execution that is common in monitoring systems. Specifically, monitoring systems read from some sensors and evaluate whether an actuation is required or not (i.e., the system is inside its normal operational envelope). If the actuation is not required the execution stops until the next periodic activation is required. If the actuation is required then such an actuation is executed to bring the system back into its operational envelope. As a result, when the task does not execute the actuation computation we have a shorter execution time (*best-execution mode*) than when it executes

the actuation (*worst-execution mode*). We call these tasks *bimodal tasks*.

If bimodal tasks have unpredictable mode switches, then the tasks need to be prepared to always execute the worst-execution mode. However, some systems can have some predictable execution ratio between the modes, i.e., it is possible to bound the minimum number of best-execution mode computations that execute for each worst-execution mode computation. We call this ratio the *modal ratio*. This is the case of the example of Figure 1, where the actuation consist of the tripping of a generator which takes the system into a degraded mode of operation. This mode keeps the system well under the emergency threshold until corrective actions are taken. It is expected that such actions or new emergencies would take longer than potential emergencies during normal operation.

A. Computation Buffering

When a bimodal task with predicable modal ratio is given enough cycles to execute its worst-execution mode in every period then we can define a bound on the minimum amount of execution time that is given to tasks but is not used during its best-execution mode. This time is available to the task during this mode. We identify this time as the *best-execution mode slack*.

The best-execution mode slack can be used to execute computations from the worst-execution mode ahead of time. For instance, we could pre-calculate values that do not depend on sensor values gathered at the instant of the activation of the worst-execution mode. In such a case, these values would be saved to be used later. We called this scheme *computation buffering*.

Computation buffering reduces the worst-case execution time reducing the overall utilization of the system. As a result, it also reduces the difference between the worst-execution and the best-execution modes and, hence, the best-execution mode slack.

When we generate more pre-computed values than required by the worst-execution mode, then we can execute the worst-execution mode and the computation buffering in a mutually-exclusive fashion. In other words, if the worst-execution mode is executed then the computation buffering is not and viceversa. This reduces (or avoids) wasted reserved computation time.

III. Decomposing Cryptographic Computation

The WACS system of Figure 1 is a system of bimodal tasks that can exhibit a predicable modal ratio. If we combine this system with stream ciphers [1] we can use computation buffering to decompose the encryption functions. In particular, stream ciphers create a keystream of the same size of the message to be encrypted (decrypted) that is later used with a simple operation (e.g. `xor`) to encrypt (decrypt) the message in a rapid fashion.

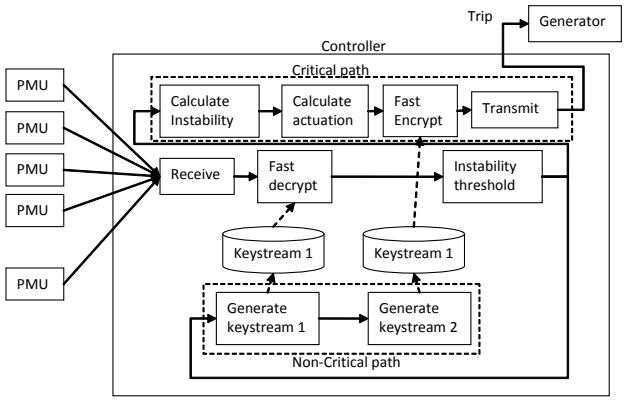


Figure 2. WACS Decomposition

The restructured system is depicted in Figure 2. We identify the two mutually-exclusive modes as two different paths: the critical path that contains the actuation functions, and the non-critical path that contains the optional computation buffering.

The new structure allows the critical path to perform a fast actuation in case a problem is detected based on the inputs from PDUs, whereas the non-critical path does not include any actuation. Such decomposition enables us to (1) reduce the total computation time of original task and (2) minimize the response time of the critical path.

The critical path of Figure 2 includes all steps from Figure 1 with encryption and decryption replaced by the fast versions. The non-critical path does not include the computation of the actuation, which frees up execution time that is used to generate the key streams. In this new structure the task is transformed into a bimodal task with execution times C_i^n for the non-critical path and C_i^c for the critical path, where both new execution times can be less than the original C_i. Furthermore, we assume that we execute C_i^c at most once per I_i executions of C_i^n. This assumtion is reasonable because (1) actuation is necessary only if the PMU measurements indicate a problem, which should be rare and (2) after an actuation the network must be given time to stabilize before the next actuation happens. The utilization of task i now observes the following relationship

$$U_i = \frac{I_i - 1}{I_i}\frac{C_i^n}{T_i} - \frac{1}{I_i}\frac{C_i^c}{T_i} = \frac{(I_i - 1)C_i^n + C_i^c}{I_i T_i} < \frac{C_i}{T_i}$$

That is, the utilization of the new bimodal task is smaller than the original one.

If we use this decomposition in our previous example it may be possible to schedule the tasks. For instance if the new execution times are:

$$C_i^n = 0.3$$
$$C_i^c = 0.35$$

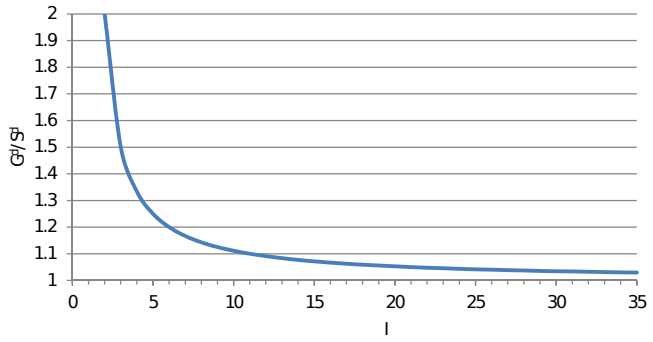


Figure 3. Needed Overproduction of Decryption Key Bits

$$I_i = 4$$

$$U = \sum_{1}^{10} \frac{(4-1)*0.3+0.35}{4*4} = 0.78125 \leq 1$$

However, in order to verify the schedulability test we need to perform a critical instant test that we will developed in Section IV. In addition, we must ensure that the keystream generation is frequent enough to produce a sufficient number of key bits such that the functions `Fast decrypt` and `Fast encrypt` can work properly. Specifically, we need to verify that $I_i - 1$ executions of the key generators produce enough key bits to decrypt at least I_i incoming PMU messages and encrypt at least one actuation message. This can be verified with the following equations

$$\begin{aligned} G_i^d(I_i - 1) - I_i S_i^d \geq 0 \\ G_i^e(I_i - 1) - S_i^e \geq 0 \end{aligned} \quad (1)$$

where:

- G_i^d is the number of bits produced by `Generate keystream 1` for the `Fast decrypt` function
- S_i^d is the number of bits consumed by the `Fast decrypt` function
- G_i^e is the number of bits produced by `Generate keystream 2` for the `Fast encrypt` function
- S_i^e is the number of bits consumed by the `Fast encrypt` function

For decryption, each round of key bit generation must produce more key bits than the length of the incoming PMU data. The ratio of generated-to-consumed bits $\frac{G_i^d}{S_i^d}$ depends on the number of executions of the `Generate keystream 1` (non-critical path) per actuation (critical path) as shown in Figure 3. It is worth noting that with an increasing modal ratio I the neecessary key stream overproduction converges toward 1 quickly.

In case of a WACS, actuations are rare compared to the number of PMU messages such that only a small number of extra key bits must be generated in each non-critial path execution. Generating the encryption key stream is even less critical because only $\frac{S_i^e}{I_i - 1}$ key bits must be generated.

We can now formally express a condition under which both new execution times are less that C_i. For this we assume that (a) the original encryption and decryption functions take the same amount of time and (b) we generate the minimum number of key bits, i.e., we use equality in relations (1). By simply adding the execution times of blocks of functionality in the two paths, we arrive at

$$\begin{aligned} C_i^n &= C_i - C_i^{\text{act}} + \text{KGEN}(G_i^d - S_i^d) + \text{KGEN}(G_i^e - S_i^e) \\ &= C_i - C_i^{\text{act}} + \text{KGEN}\left(\frac{1}{I_i - 1} S_i^d - \frac{I_i - 2}{I_i - 1} S_i^e\right) \end{aligned}$$

and

$$C_i^c = C_i - \text{KGEN}(S_i^d + S_i^e) \leq C_i$$

where C_i^{act} is the time needed to calculate a necessary actuation and $\text{KGEN}(n)$ is the execution time to generate n bits of a key stream. C_i^{act} includes the functions `Calculate Instability`, `Calculate Actuation`, `Fast Encrypt`, and `Transmit` from Figure 2.

It follows that $C_i^n \leq C_i$ if

$$C_i^{\text{act}} > \text{KGEN}\left(\frac{1}{I_i - 1} S_i^d - \frac{I_i - 2}{I_i - 1} S_i^e\right)$$

This condition puts an upper bound on the time to generate additional key bits needed for decryption (and therefore the length of incoming PMU messages) when the critical execution path is taken.

IV. New Response-Time Test

In order to evaluate the schedulability of a taskset we use the response-time test using the multiframe task model presented in [2]. With this model if we ensure that we structure each task as a multiframe task with the following structure

$$\tau_i = \begin{cases} (\langle C_{i,1}^n, \ldots, C_{i,I_i-1}^n, C_i^c \rangle, T_i) & \text{if } C_i^n > C_i^c \\ (\langle C_i^c, C_{i,1}^n, \ldots, C_{i,I_i-1}^n \rangle, T_i) & \text{otherwise} \end{cases}$$

where we have $I_i - 1$ periodic executions of the non-critical path followed by an execution of the critical path if the non-critical path execution is larger than the critical execution, or one critical execution followed by $I_i - 1$ non-critical executions otherwise. We can now evaluate its schedulability with the traditional rate-monotonic response-time tests. In particular, in the critical instance of this task set all tasks request execution at the same time, and the task set is schedulable when it is schedulable in the critical instance [2].

We use these results to evaluate the response time of task τ_i. We can use the following recurrence equations and execute them until they converge ($R^{r+1} = R^r$) to calculate

the precise response time of the task ($R_{i,c}$ for the critical path and $R_{i,n}$ for the non-critical path).

$$R_{i,n}^0 = C_i^n$$

$$R_{i,n}^{r+1} = C_i^n + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i^r}{T_j} \right\rceil C_j^n + \text{CP}_i(j) \right)$$

$$R_{i,c}^0 = C_i^c$$

$$R_{i,c}^{r+1} = C_i^c + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i^r}{T_j} \right\rceil C_j^n + \text{CP}_i(j) \right)$$

where $CP_i(j)$ is the preemption from the critical path defined as:

$$\text{CP}_i(j) = \begin{cases} \left\lceil \frac{R_i^r}{T_j I_j} \right\rceil (C_j^c - C_j^n) & \text{if } C_j^c > C_j^n \\ - \left\lfloor \frac{R_i^r}{T_j I_j} \right\rfloor (C_j^n - C_j^c) & \text{otherwise} \end{cases}$$

and $hp(i)$ is the set of all the indices of the tasks with shorter or the same period excluding task i (in our example $j \neq i$). This set represents the tasks with higher priority than i when rate-monotonic priorities are used. Once the response time is calculated we only need to verify that it is smaller than its period.

$$R_{i,c}^r \leq T_i$$

$$R_{i,n}^r \leq T_i$$

For our example the response time test for the critical path of our tasks would be as follows:

$$R_{1,c}^0 = 0.35$$

$$R_{1,c}^1 = 0.35 + \sum_1^9 \left\lceil \frac{0.35}{4*4} \right\rceil 0.05 + \sum_1^9 \left\lceil \frac{0.35}{4} \right\rceil 0.3 = 3.5$$

$$R_{1,c}^2 = 0.35 + \sum_1^9 \left\lceil \frac{3.5}{4*4} \right\rceil 0.05 + \sum_1^9 \left\lceil \frac{3.5}{4} \right\rceil 0.3 = 3.5$$

This means that the critical path of the task will finish in less than $4ms$. For the non-critical path we get

$$R_{1,n}^0 = 0.3$$

$$R_{1,n}^1 = 0.3 + \sum_1^9 \left\lceil \frac{0.3}{4*4} \right\rceil 0.05 + \sum_1^9 \left\lceil \frac{0.3}{4} \right\rceil 0.3 = 3.45$$

$$R_{1,n}^2 = 0.3 + \sum_1^9 \left\lceil \frac{3.45}{4*4} \right\rceil 0.05 + \sum_1^9 \left\lceil \frac{3.45}{4} \right\rceil 0.3 = 3.45$$

such that the non-critical path also finishes in less than $4ms$ and the system is schedulable.

V. Concluding Remarks

In this paper we presented a new task model and scheduling for bimodal tasks with computation buffering and predictable modal ratio. This model takes advantage of the fact that the trailing part of a task is not always executed (e.g. the actuation). When this happens our tasks pre-calculate and save values to be used by the trailing part of the task in a future activation. These values, in turn, reduce the execution time of the trailing part. We showed how this novel structure allows us to both reduce the utilization of the task (and the taskset) and reduce the response time of the trailing part of the task. We also presented the mapping of this task to the multi-frame scheduling model and the modifications to the response time calculation. Finally, we used an example from the smart grid throughout the paper to illustrate the benefits or our approach.

References

[1] A.J.Menezes, C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. 1997.

[2] A.K.Mok and D.Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10), 1997.

[3] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Systems*, 17:5–22, 1999. 10.1023/A:1008030427220.

[4] Sanjoy K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24:93–128, 2003. 10.1023/A:1021711220939.

[5] II E.O. Schwitzer, D. Whitehead, A. Guzman, Y.Gong, and M.Donolo. Advanced real-time synchrophasor applications. In *Proceedings of the 35th, Annual Western Protective Relay Conference*, 2008.

[6] Energy Sector Control Systems Group. Roadmap to secure energy delivery systems, 2011.

[7] Shujun Li, Xuan Zheng, Xuanqin Mou, and Yuanlong Cai. The digraph real-time task model. In *In Proceedings of the VI SPIE Real-Time Imaging*, volume 4666, pages 149–160, 2002.

[8] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20:46–61, January 1973.

[9] G.A. Spanos and T.B. Maples. Security for real-time mpeg compressed video in distributed multimedia applications. In *Computers and Communications, 1996., Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on*, pages 72 –78, March 1996.

[10] M. Stigge, P. Ekberg, Nan Guan, and Wang Yi. The digraph real-time task model. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 71–80, April 2011.

Notes

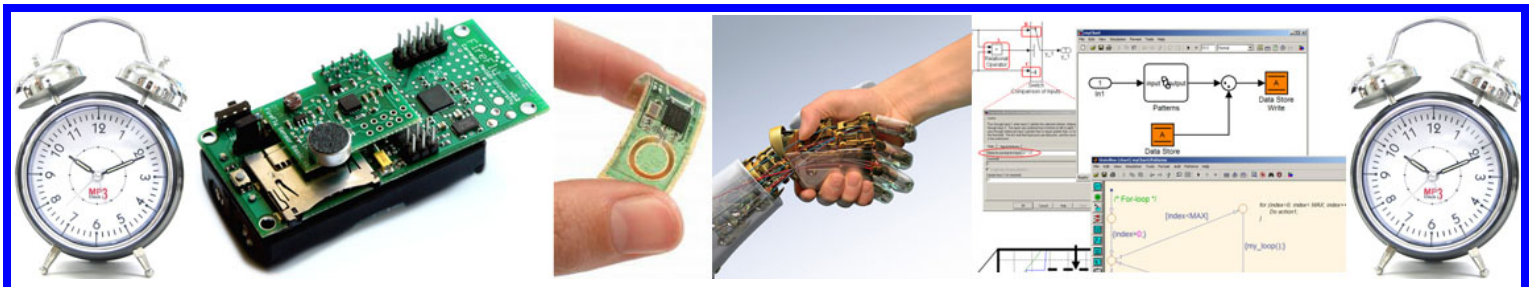
Notes

Notes

Notes

Notes

Notes



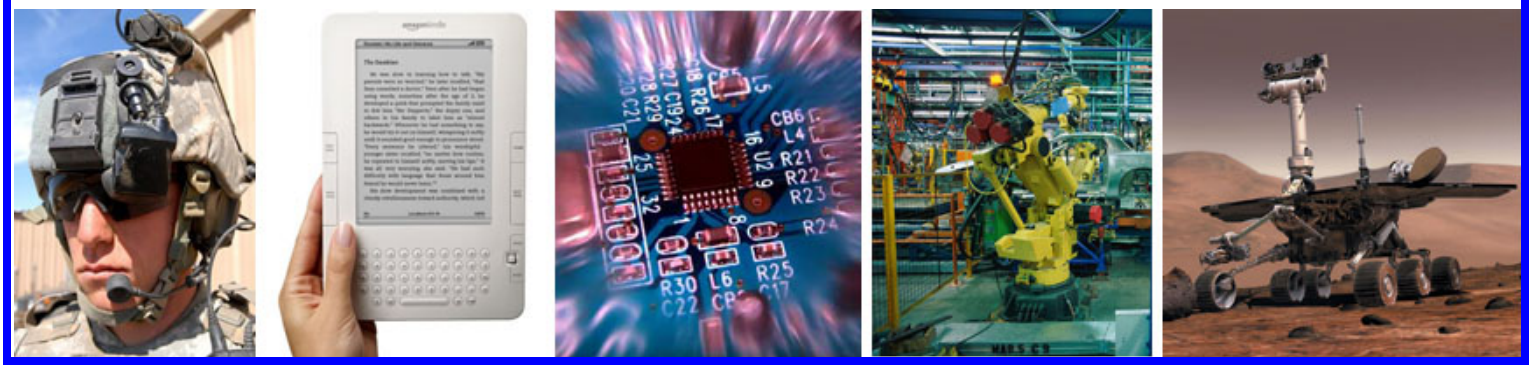
2nd Workshop on

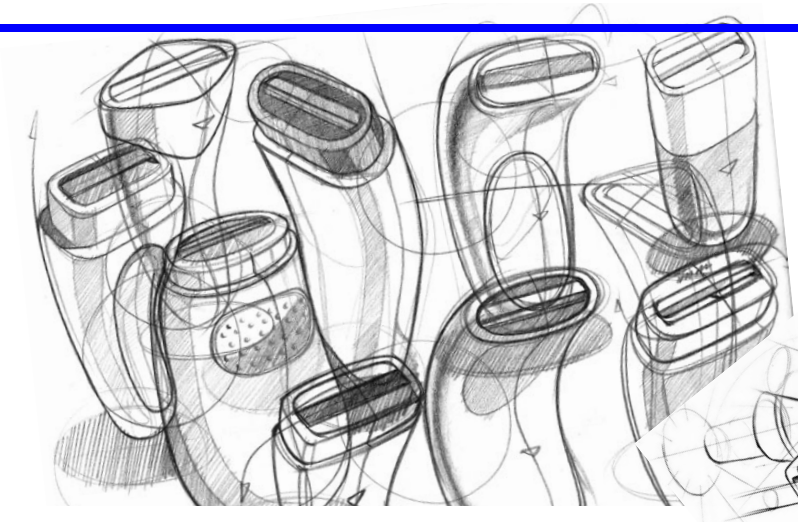
Analytic Virtual Integration for Cyber-Physical Systems

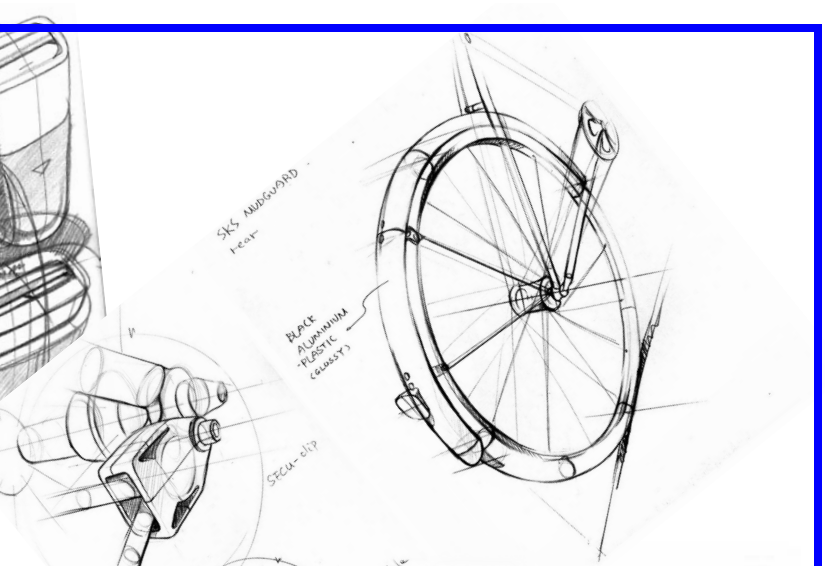
IEEE AVICPS 2011

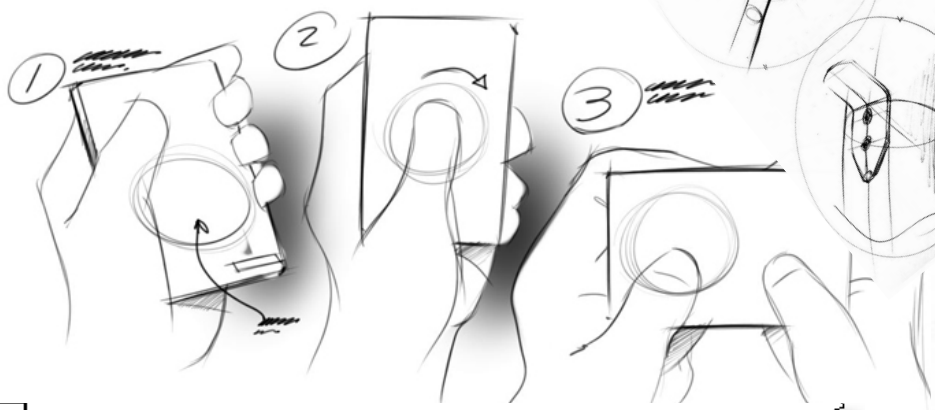
In conjunction with IEEE RTSS, Vienna, Austria

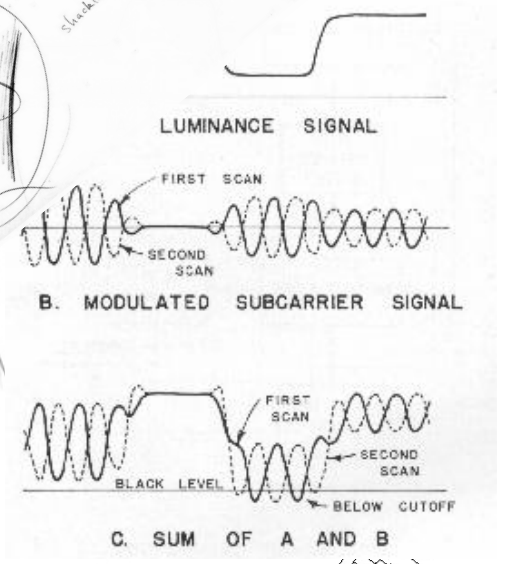
November 29th, 2011

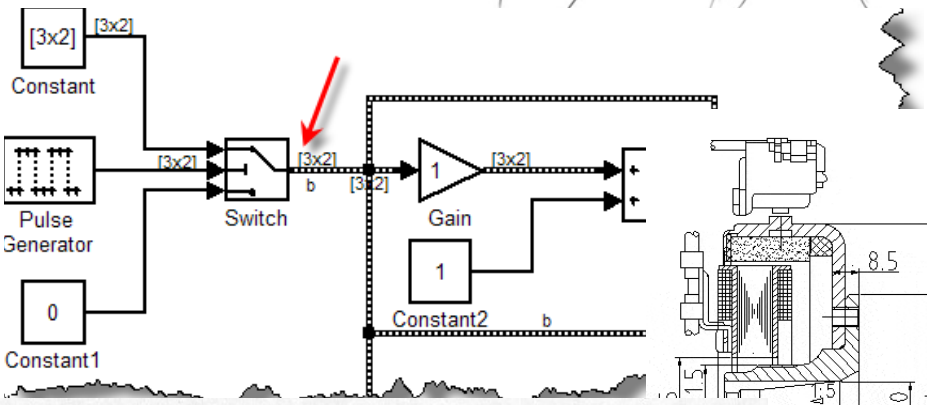


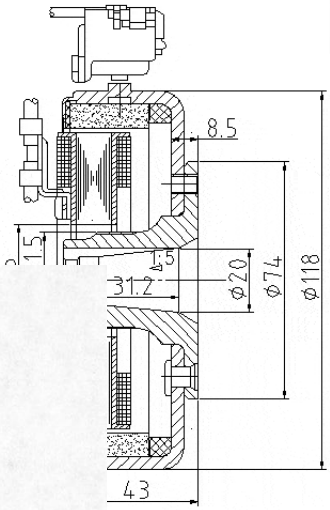


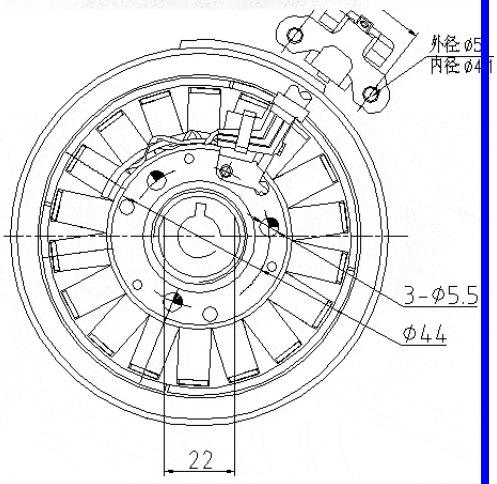


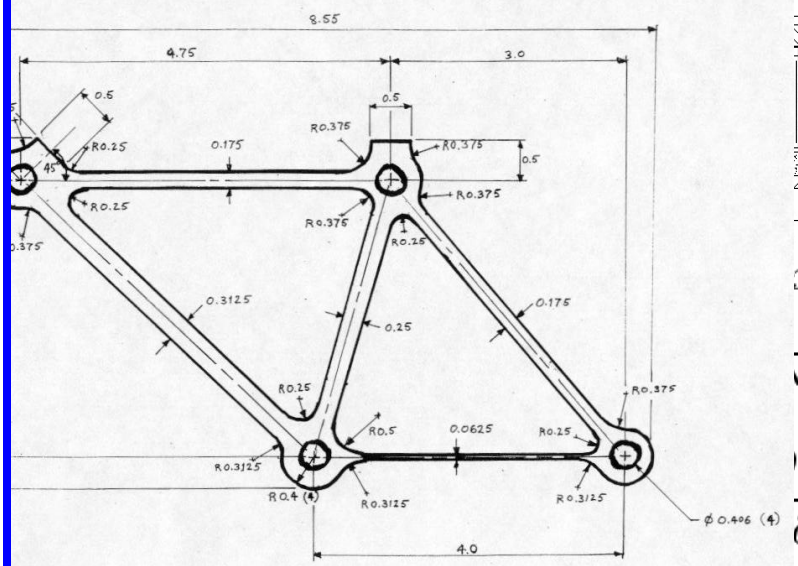












$$x_i \qquad c\,\partial x_k\,\partial t \qquad c^2\ \partial t^2 \qquad c$$

$$\nabla^2 A_k + \frac{1}{c}\frac{\partial}{\partial x_k}\frac{\partial \phi}{\partial t} + \frac{1}{c^2}\frac{\partial^2 A_k}{\partial t^2} = \frac{4\pi}{c} J$$

$$\frac{\partial^2 A_k}{\partial t^2} + \frac{\partial}{\partial x_k}\left(\vec{\nabla}\cdot\vec{A} + \frac{1}{c}\frac{\partial \phi}{\partial t}\right) = \frac{4\pi}{c} J$$

$$\frac{1}{c^2}\frac{\partial^2 \vec{A}}{\partial t^2} + \vec{\nabla}\left(\vec{\nabla}\cdot\vec{A} + \frac{1}{c}\frac{\partial \phi}{\partial t}\right) = \frac{4\pi}{c}$$