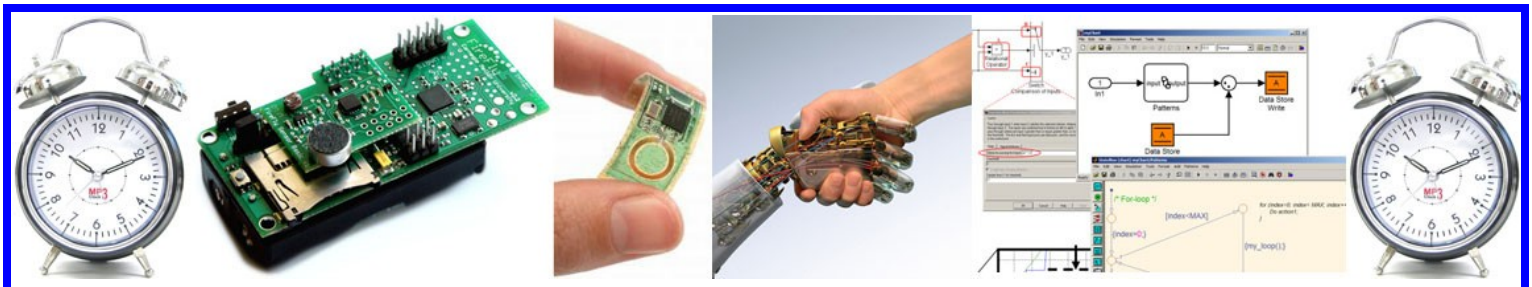


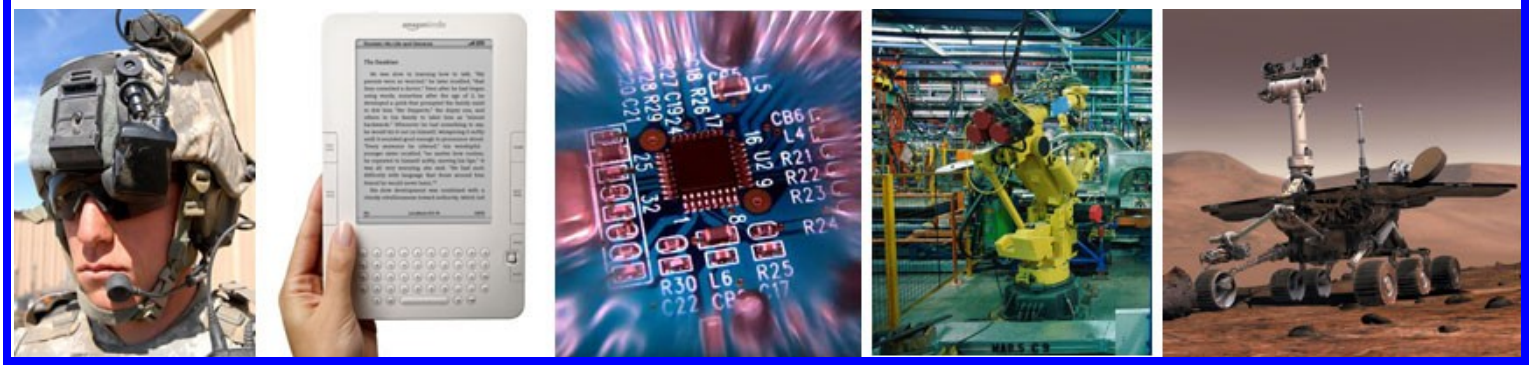
3rd Workshop on

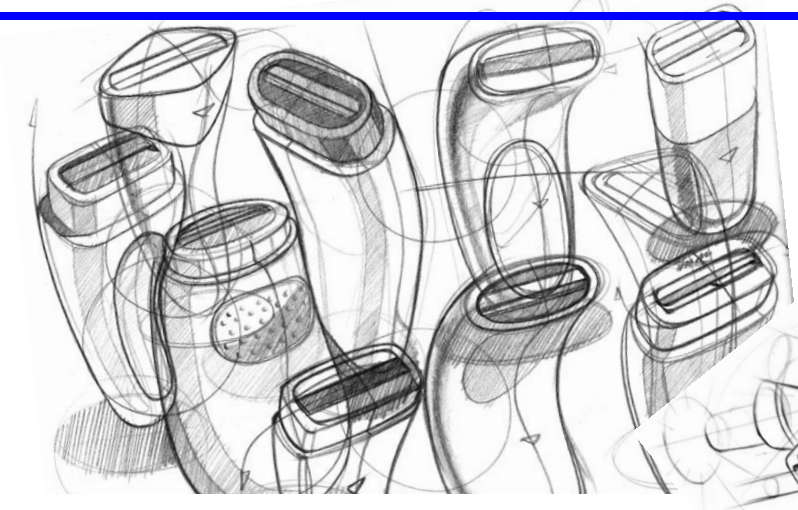
Analytic Virtual Integration for Cyber-Physical Systems

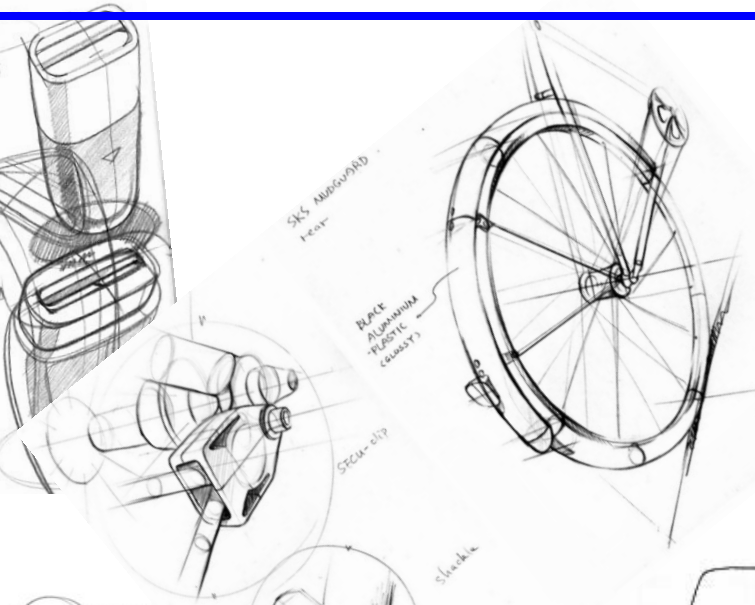
AVICPS 2012

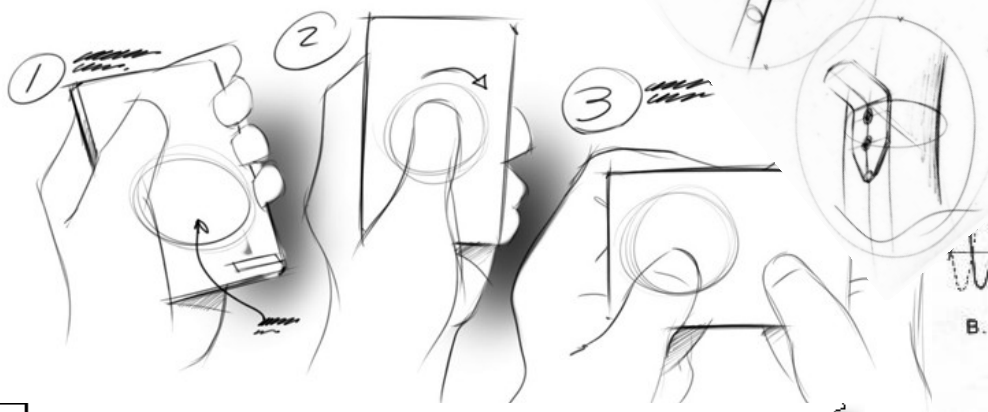
In conjunction with IEEE RTSS, San Juan, Puerto Rico

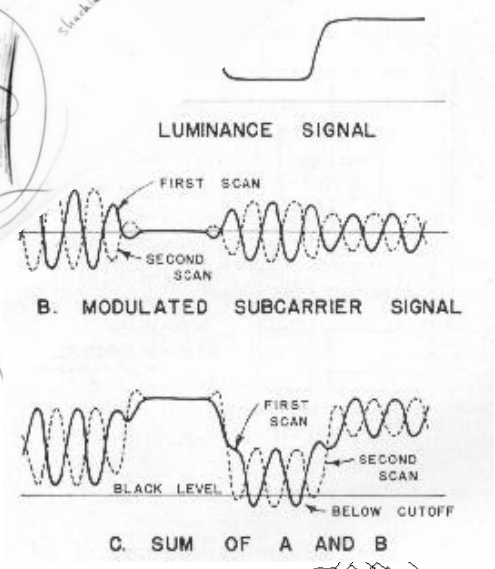
December 4th, 2012

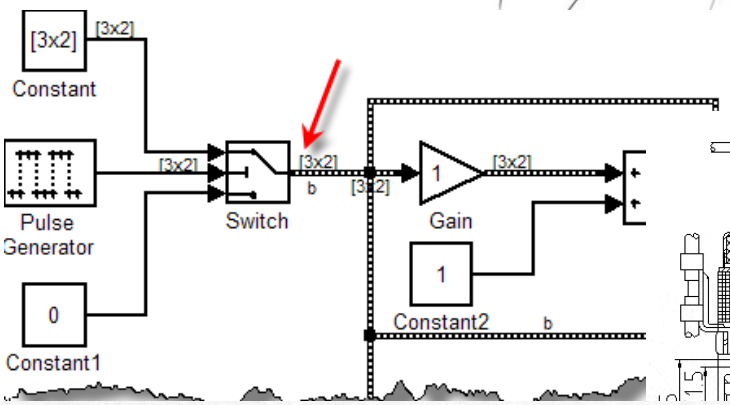


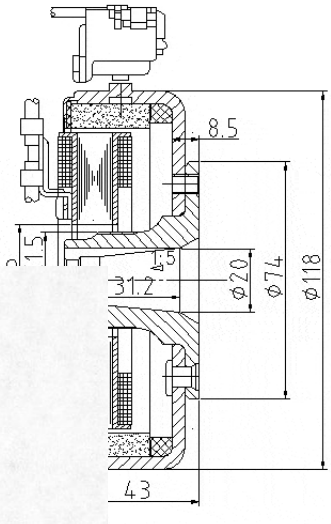


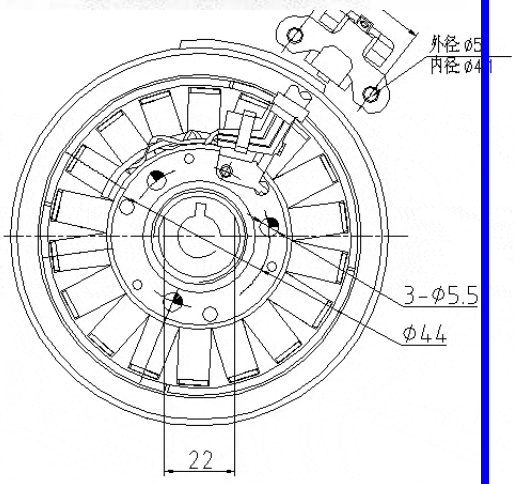


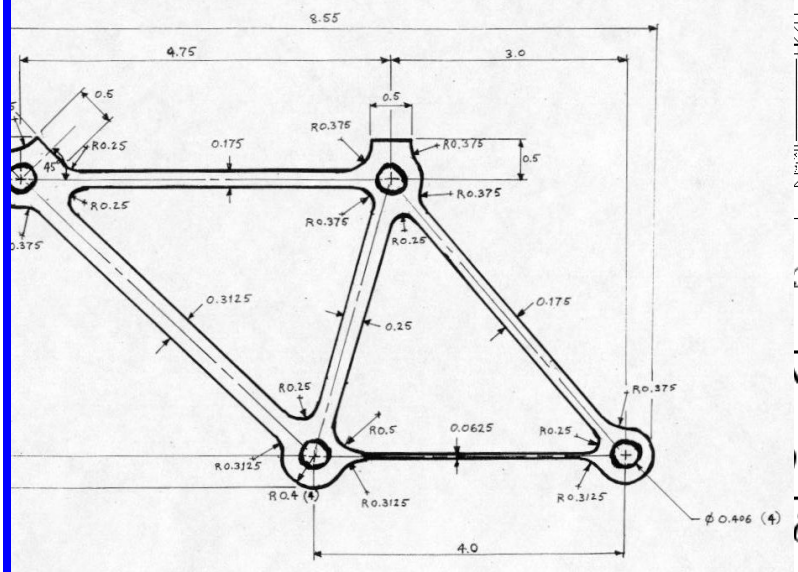












NOTES
1. ALL DIMENSIONS IN INCHES.
2. THICKNESS 0.25"
3. MATERIAL: AL 6061-T6

ACROBAT
SCALE: 1":1"

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

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

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

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

Chair's Foreword

Welcome to AVICPS 2012, the 3rd Workshop on Analytical Virtual Integration of Cyber-Physical Systems.

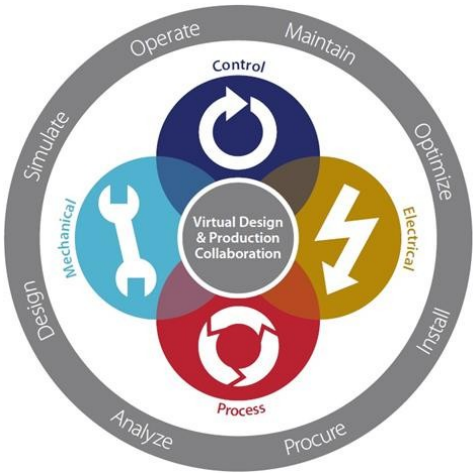
The goal of the workshop is to explore architectural design patterns, tools and the theoretical analytical foundations for creating common system-wide composition models, where key properties can be studied and guarantees provided before the start of actual development. The concept of virtual integration has emerged with the advent of model-based development. Models allow us to reason about the system and predict its properties before the system is built. In this way, we can discover design flaws early in the development process, when errors are easier and cheaper to fix. A large fraction of errors in the system design emerge during integration, therefore analyzing integration at the modeling level is of particular importance.

Cyber-physical systems (CPS) bring an additional level of complexity to system in general, and to the problem of integration in particular. CPS components may affect each other through digital communication channels, as in any other computer-based system, but also by coupling through the physical world. Physical influence on the digital communication channels has to be considered as well.

This year, AVICPS brings together researchers addressing multiple aspects of virtual integration. The workshop includes papers on various modeling approaches for CPS components and system environments, architecture-driven system development, safety analysis, among others. Presentations of contributed papers are supplemented by two keynote talks by Peter Feiler (Software Engineering Institute) and Michael Whalen (University of Minnesota), who bring complementary perspectives on the area.

As in the past, AVICPS is held together with the Real-Time Systems Symposium, which keeps us in touch with the larger community of researchers exploring related problems. We are grateful to RTSS organizers for providing this opportunity and handling many essential aspects of the workshop organization. We also acknowledge generous support of the Software Engineering Institute and PRECISE Center at the University of Pennsylvania. Finally, our sincere thanks go to the program committee members. This workshop would not be possible without their time and effort. We hope you will enjoy participating in AVICPS 2012, and making it a success!

Sagar Chaki and Oleg Sokolsky
AVICPS 2012 co-Chairs



Program Committee:

Rajeev Alur, University of Pennsylvania, USA
Saddek Bensalem, Verimag, France
Ken Butts, Toyota, USA
David Broman, UC Berkeley, USA
Julien Delange, European Space Agency, Netherlands
Jorgen Hansson, Chalmers University, Sweden
Boudewijn Haverkort, Embedded Systems Institute, Netherlands
Jerome Hugues, Institute for Space and Aeronautics Engineering, France
Mirko Jakovljevic, TTTech, USA
Christoph Kirsch, University of Salzburg, Austria
Jens Knoop, TU Vienna, Austria
Bruce Krogh, Carnegie Mellon University, USA
Thomas Noll, RWTH Aachen, Germany
Roman Obermaisser, University of Siegen, Germany
K. C. Shashidhar, Mathworks, USA
Sandeep Shukla, Virginia Tech, USA
Mike Whalen, University of Minnesota, USA

Program Co-Chairs:

Oleg Sokolsky, University of Pennsylvania (Penn), USA
Sagar Chaki, Software Engineering Institute (SEI), USA

Workshop Agenda

8:00 - 8:45am: Registration and Coffee

8:45 - 9:00am: Welcome and Kick-off

9:00 - 10:00am: Session #1

Keynote #1: Analytical Architecture Fault Models (1 hr)
Dr. Peter H. Feiler, Software Engineering Institute

10.00 - 10:30am: Coffee break

10:30 - 12:00pm: Session #2

A Simulation Framework for Design of Mixed Time/Event-Triggered Distributed Control Systems with SystemC/TLM (25 mins)
Zhenkai Zhang (zhenkai.zhang@vanderbilt.edu), Joseph Porter (jporter@isis.vanderbilt.edu), Xenofon Koutsoukos (xenofon.koutsoukos@vanderbilt.edu), Janos Sztipanovits (janos.sztipanovits@vanderbilt.edu)

An Analytical Model of the CAN Bus for Online Schedulability Test (25 mins)
Zhenwu Shi (zwshi@gatech.edu), Fumin Zhang (fumin@gatech.edu)

Analytic Certification Technologies for Military Avionics (20 mins)
Russell Kegley (russell.b.kegley@lmco.com), Jonathan Preston (jonathan.d.preston@lmco.com)

Integration of Mixed-Criticality Cyber-Physical Systems with Criticality Layers (20 mins)
Dionisio de Niz (dionisio@sei.cmu.edu), Anthony Rowe (agr@ece.cmu.edu)

12:00 - 1:00pm: Lunch

1:00 - 2:30pm: Session #3

Keynote #2: Compositional Safety and Security Analysis of Architecture Models (1 hr)
Dr. Michael W. Whalen, University of Minnesota

Simulation-Based Design Verification of Real-Time Distributed Automotive Systems (25 mins)
Shin'Ichi Shiraishi (sshiraishi@us.toyota-itc.com)

2:30 - 3:00pm: Coffee break

3:00 - 4:30pm: Session #4

Online Construction of Analytical Prediction Models for Physical Environments: Application to Traffic Scene Modeling (25 mins)
Anurag Umbarkar (aumbarka@ic.sunysb.edu), Shreyas Kodasara (sk.shreyas@gmail.com), Alex Doboli (adoboli@ece.sunysb.edu)

Towards a Model-Driven Engineering Software Development Framework (20 mins)
Julien Delange (julien.delange@gmail.com), Maxime Perrotin (Maxime.Perrotin@esa.int), Samir Bennani (samir.bennani@esa.int)

Roundtable Discussion (40 mins)

Closing Remarks (5 mins)

Keynote Speaker 1

Software Engineering Institute
Carnegie Mellon

Dr. Peter H. Feiler, Software Engineering Institute

Title: Analytical Architecture Fault Models

Abstract: *In this talk we summarize challenges in safety-critical software-intensive systems and introduce the concept of an analyzable architecture fault model expressed in SAE AADL and its Error Model Annex standard. We then present its use early in the development life cycle supporting hazard and fault impact analysis, show its ability to provide compositional analysis, discuss the interaction between operational and failure modes, and conclude with an illustration of its use to gain better understanding of the intricacies of desired timing behavior in safety-critical systems.*

Bio: Peter Feiler is a 27 year veteran and currently a senior member of the Research, Technology, and Systems Solutions (RTSS) program of the Software Engineering Institute (SEI). His current research interest is in improving the quality of safety-critical software-intensive systems, aka. cyber-physical systems, through architecture-centric virtual integration and analysis throughout the development life cycle to complement traditional testing resulting in major reduction in rework and qualification costs. Peter Feiler has been the technical lead and main author of the SAE Architecture Analysis & Design Language (AADL) standard. He has a Ph.D. in Computer Science from Carnegie Mellon.

NOTES

NOTES

Keynote Speaker 2

Dr. Michael W. Whalen, University of Minnesota

Title: Compositional Safety and Security Analysis of Architecture Models

Abstract: *This talk presents a design flow and supporting analysis tools for compositional analysis of system architectures. We focus on system architecture models composed from libraries of components and complexity-reducing design patterns with formally verified properties. This allows new system designs to be developed rapidly using patterns that have been shown to reduce unnecessary complexity and coupling between components. Components and patterns are annotated with formal contracts describing their guaranteed behaviors and the contextual assumptions that must be satisfied for their correct operation. We describe the compositional reasoning framework that we have developed for proving the correctness of a system design, and illustrate it with an example based on an aircraft flight control system.*

Bio: Dr. Michael Whalen is the Program Director at the University of Minnesota Software Engineering Center. Dr. Whalen is interested in formal analysis, language translation, testing, and requirements engineering. He has developed simulation, translation, testing, and formal analysis tools for Model-Based Development languages including Simulink, Stateflow, SCADE, and RSML-e, and has published more than 30 papers on these topics. He has led successful formal verification projects on large industrial avionics models, including displays (Rockwell-Collins ADGS-2100 Window Manager), redundancy management and control allocation (AFRL CerTA FCS program) and autoland (AFRL CerTA CPD program).

NOTES

NOTES

Abstracts:

A Simulation Framework for Design of Mixed Time/Event-Triggered Distributed Control Systems with SystemC/TLM

Zhenkai Zhang (zhenkai.zhang@vanderbilt.edu), Joseph Porter (jporter@isis.vanderbilt.edu), Xenofon Koutsoukos (xenofon.koutsoukos@vanderbilt.edu), Janos Sztipanovits (janos.sztipanovits@vanderbilt.edu)

Mixed time/event-triggered (TT/ET) distributed control systems are complex systems which have emerged in many cyber-physical domains but have been difficult to evaluate at early design stages. In order to reveal design flaws as early as possible, this paper proposes a simulation framework based on an executable virtual platform model in SystemC/TLM. The executable platform is generated using a model-based approach from a system designed in the Embedded Systems Modeling Language (ESMoL). The virtual platform consists of three types of abstract models, the RTOS model, the communication system model, and the hardware model, to capture different behaviors of the mixed TT/ET distributed control systems. Preliminary results from a case study using a Quadrotor flight control system are
used to illustrate the approach.

An Analytical Model of the CAN Bus for Online Schedulability Test

Zhenwu Shi (zwshi@gatech.edu), Fumin Zhang (fumin@gatech.edu)

Controller area network (CAN) is a prioritybased bus that supports real-time communication. Existing
schedulability analysis for the CAN bus is peformed at the design stage, by assuming that all message information is known in advance. However, in pratice, the CAN bus may run in a dynamic environment, where complete specifications may not be available at the design stage and operational requirements may change at system run-time. In this paper, we develop an analytical model that describes the dynamics of message transmission on the CAN bus. Based on this analytical timing model, we then propose an online test that effectively checks the schedulability of the CAN bus, in the presence of online adjustments of message streams. Simulations show that the online test can accurately report the loss of scheduability on the CAN bus.

Analytic Certification Technologies for Military Avionics

Russell Kegley (russell.b.kegley@lmco.com), Jonathan Preston (jonathan.d.preston@lmco.com)

Historic approaches to upgrading the capabilities of military aircraft by inserting new technologies have become so costly that warfighters may be forced to operate with less than current technology can deliver. Even inserting new upgrades alongside the legacy systems can be very expensive if it requires large modifications to legacy software, triggering extensive retest. A way to meet this challenge is to insert upgrades alongside legacy systems, virtually replacing old capabilities with new while leaving the old software in place. This approach carries with it new challenges which might be met with analytic innovations.

Integration of Mixed-Criticality Cyber-Physical Systems with Criticality Layers

Dionisio de Niz (dionisio@sei.cmu.edu), Anthony Rowe (agr@ece.cmu.edu)

Large Cyber-Physical Systems such as avionics and automotive systems often require large integration efforts etween third-party components. These components provide functionality at different levels of criticality yet share many of the same underlying resources (CPU, Memory, Network, Disk, Transducers). As a result, protections mechanisms are needed to prevent lower-criticality task from interfering with higher-criticality ones. In this paper, we discuss how traditional temporal protection mechanisms such ARIC 653 partitions fail to fully protect high-criticality tasks

from lower-criticality ones. We then show how the Zero-Slack QRAM scheduler (ZSQRAM) can be used in multi-layer systems to avoid these problems. Furthermore, we propose the use of criticality layers based on the asymmetric protection scheme of ZS-QRAM in order to simplify this integration, increase its robustness, and reduce its resource usage. Finally, we discuss some open issues that need to be addressed in order to remove some of the limitations of this approach.

Simulation-Based Design Verification of Real-Time Distributed Automotive Systems

Shin'Ichi Shiraishi (sshiraishi@us.toyota-itc.com)

In this paper, we propose a simulation-based verification technique for real-time distributed automotive systems. The proposed technique enables accurate simulation and it utilizes only limited information that can be collected in the design phases of development. In other words, the proposed technique enables the design verification of automotive systems. Therefore, the proposed design verification during early phases of development can potentially enhance the productivity of automotive systems. Moreover, the proposed technique is developed by extending the modeling fundamentals of a single commercial tool known as OPNET Modeler. This tool is widely used in the network technology domain, and it has been provided with sufficient technical support by its vendor. Thus, the proposed method is now available for immediate application to real-world automotive system development.

Online Construction of Analytical Prediction Models for Physical Environments: Application to Traffic Scene Modeling

Anurag Umbarkar (aumbarka@ic.sunysb.edu), Shreyas Kodasara (sk.shreyas@gmail.com), Alex Doboli (adoboli@ece.sunysb.edu)

This paper presents a methodology to model the dynamics of traffic scenes, including the participating vehicles, vehicle clusters, the attributes and relations of the scene elements, and related events, like cluster merging and splitting. Compared to other methods, this methodology constructs the models online using data coming from sensors. The main steps are to identify the elements of a scene, to find the relations among the elements, and to construct analytical prediction models for the traffic scene dynamics. The paper discusses all the related theoretical aspects, including ontologies for traffic scene description, stochastic prediction of event sequences, and vehicle and cluster identification using sound-based vehicle localization.

Towards a Model-Driven Engineering Software Development Framework

Julien Delange (julien.delange@gmail.com), Maxime Perrotin (Maxime.Perrotin@esa.int), Samir Bennani (samir.bennani@esa.int)

Design and Implementation of Safety-Critical Systems is becoming very difficult becauses it involves many
requirements coming from different engineering domains. Due to the increase of complexity, software of such systems can no longer be produced with traditional methods, which show their limit over time. In that context, new development approaches have to be introduced to avoid actual development traps and pitfalls. Among them, the Model-Driven Engineering approach consists at representing system artifacts with models and autogenerate the code by refining them from high-level concepts down to the code. However, as for every new approach, it also brings new problems such as requirements consistency among the different notations (models) as well as integration issues (for example, making sure that implementation code from different models will behave correctly when merged on a single execution platform).

This article presents our experience for integrating Guidance and Navigation Control (GNC) algorithms designed with Application Models (Simulink) with Architecture Models (AADL). The process relies on code generator for both models and integrate it on a typical execution platform. In particular, we focus on the challenges of the integration, illustrating the practical problems we faced for producing a space system using a Model-Driven Engineering Approach.

NOTES

NOTES

NOTES

Simulation-Based Design Verification of Real-Time Distributed Automotive Systems

Shin'ichi SHIRAISHI

TOYOTA InfoTechnology Center, U.S.A., Inc.
465 Bernardo Avenue
Mountain View
CA 94043
Email: sshiraishi@us.toyota-itc.com

***Abstract*—In this paper, we propose a simulation-based verification technique for real-time distributed automotive systems. The proposed technique enables accurate simulation and it utilizes only limited information that can be collected in the design phases of development. In other words, the proposed technique enables the design verification of automotive systems. Therefore, the proposed design verification during early phases of development can potentially enhance the productivity of automotive systems. Moreover, the proposed technique is developed by extending the modeling fundamentals of a single commercial tool known as OPNET Modeler. This tool is widely used in the network technology domain, and it has been provided with sufficient technical support by its vendor. Thus, the proposed method is now available for immediate application to real-world automotive system development.**

***Keywords*-Automotive Systems, System Design Verification, Simulation, Real-Time Systems, Controller Area Network, OPNET Modeler**

I. Introduction

Automotive systems are highly complex distributed embedded systems. For example, a certain luxurious car employs several automotive systems constructed on the basis of complex networks wherein nearly 100 electronic control units (ECUs) are interconnected. In order to efficiently develop such complex systems, system design verification is required during the early phases of development. Moreover, the upcoming automotive safety standard (ISO 26262 [1]) stipulates that design phase verification is required to ensure high-level functional safety.

In spite of such a high demand for early-phase verification in real-world automotive system development, there is no straightforward approach to design verification. This drawback is particularly critical in software development; hence, early-phase verification before the implementation phase is not possible without prototyping. Unfortunately, existing techniques cannot simultaneously deal with the various aspects of automotive systems, such as, (1) real-time systems, (2) complex distributed systems based on networks, and (3) cost-sensitive system development.

Regarding the first aspect, most automotive systems are real-time systems; an accurate analysis is required in terms of the timeliness. The timeliness of automotive systems is usually verified by using cycle-accurate simulation techniques with instruction set simulators. This type of simulation is generally known as *virtual prototyping*, and it requires software implementation, i.e., a set of source code. Although virtual prototyping allows us to avoid hardware prototyping, it does not enable the design-phase verification of software. Virtualization can also be achieved using *virtual platforms*. Virtual platforms enable us to implement software without any RTL descriptions of its execution platform (hardware); however, it requires a set of source code.

The second aspect implies that most automotive systems are highly distributed systems based on multiple interconnected ECUs. In other words, for design verification, it is not sufficient to consider only the software used in automotive systems, and its execution platform (hardware part), including microcontrollers, RAMs, and ROMs. Such hardware and software components are the main verification objectives in simulations based on virtual prototyping or virtual platforms; however, the network component of automotive systems is beyond their scope. Thus, we need an end-to-end verification technique for the network component, which interconnects the ECUs, hardware, and software.

The third aspect reflects the strict cost constraint on automotive system development. This implies that car manufacturers (OEMs) cannot afford to implement all the necessary development techniques as in-house tools. Thus, we need to effectively use commercial tools that support early-phase verification, and exploit technical support from their vendors. A recent paper [2] proposes a holistic simulation technique for automotive systems; however, its pragmatic application is unlikely because it does not offer any tool implementation. Another recent paper [3] presents a co-simulation technique based on multiple simulation tools. Despite the advantages of such techniques, they are not considered practical from the viewpoint of the purchase and maintenance cost of multiple commercial tools.

Therefore, in this paper, we try to overcome the aforementioned drawbacks and develop a novel design verification technique for automotive systems using a single

commercial tool. More precisely, we propose a high-level modeling approach to the holistic simulation of real-time distributed automotive systems. The proposed technique can simultaneously simulate the detailed behavior of the three types of components: software, hardware, and networks; it utilizes only the limited information available in the design phases. Thus, the proposed simulation technique enables several end-to-end analyses that are essential for the verification of real-time systems. In addition, we can evaluate the performance of systems under development in their design phases. Such early performance estimation facilitates system design verification; hence, the proposed technique can potentially enhance the quality and productivity of automotive systems. In addition, the proposed technique depends only on a single commercial tool that is widely used in the network technology domain. Therefore, this technique has the potential for rapid deployment among stakeholders involved in automotive system development.

II. Real-World Example: Real-Time Distributed Automotive Systems

In this section, we present a real-world example of complex automotive systems. Through this case study, we clarify the difficulties related to the design verification of automotive systems.

A. Automotive Systems Based on Complex In-Vehicle Networks

Figure 1 shows an in-vehicle controller area network (CAN) [4] of a certain luxurious car in the Japanese market. Three CAN buses are interconnected via a gateway ECU (hereafter, referred to as a GW ECU), and each CAN bus has 10–20 ECUs. Over this large-scale network, more than 200 types of messages are exchanged. The network shown in Fig. 1 is a part of the entire structure of networks in the sampled car. In general, luxury-class cars have approximately 100 ECUs that are interconnected by 10 buses such as CAN and LIN (local interconnect network) [5] buses. This implies that besides the GW ECU shown in Fig. 1, luxurious cars have several types of GW ECUs with different specifications. For example, one GW ECU bridges different speed buses such as 250 kbps and 500 kbps; another GW ECU bridges different protocols such as CAN and LIN, and so on.

Networks in which several ECUs are interconnected serve as platforms that support various real-time automotive systems such as adaptive cruise control systems, vehicle dynamics integrated management systems, and pre-crush safety systems. For example, an adaptive cruise control system requires an engine control ECU, a skid control ECU, and a radar sensor ECU; these ECUs must be interconnected via a network. Simultaneously, the pre-crush safety system utilizes the same components. Similarly, numerous components are shared among several different systems; thus, networks

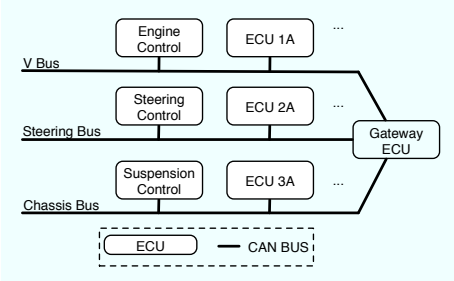


Figure 1. Distributed Automotive Systems (CAN-based Network).

and GW ECUs are typical components shared by multiple systems.

To summarize, a new end-to-end analysis technique is required for shared components such as networks and GW ECUs for the purpose of system design verification, which is required by the ISO 26262 standard.

B. Architecture of CAN–CAN Gateway ECU

When we discuss design verification of automotive systems, we need to focus on the fact that GW ECUs are important components for system design verification. As mentioned earlier, most automotive systems are real-time systems; hence, we need to analyze the timeliness of such systems in an end-to-end manner. On the other hand, delays produced by GW ECUs have a significant impact on end-to-end analysis; however, the early estimation of end-to-end delays is complicated owing to the complex functionalities of GW ECUs. Paradoxically, this means that GW ECUs are ideal subjects for discussing design verification in detail; hence, they are central to this paper, hereafter. It should be noted that the following discussion can be applied to other types of ECUs.

Figure 2 shows the architecture of the GW ECU shown in Fig. 1. This figure indicates that the GW ECU is equipped with priority control in order to realize end-to-end QoS management. Moreover, a computation-intensive functionality, i.e., routing, is realized by hardware, whereas the others are implemented by software. This design decision is motivated by a trade-off between the latency and the flexibility of the GW ECU. This implies that design verification must be a co-verification of hardware and software. In addition to the hardware and software components, some buffers are located at the borders among hardware, software, and networks. The behavior analysis of these buffers, such as a buffer overflow or underflow, is critical to design verification.

III. Potential Approaches to System Verification and their Drawbacks

There exist two potential approaches to system design verification: an analytical (formal) approach and a simulation-based approach. The analytical approach is effective in investigating corner cases. If we apply the analytical approach

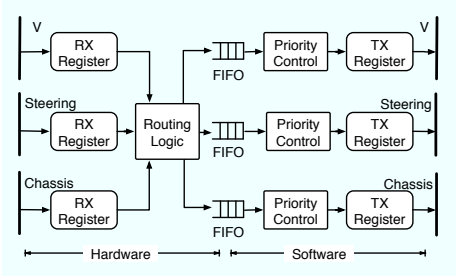


Figure 2. Architecture of Multi-Channel CAN Gateway ECU.

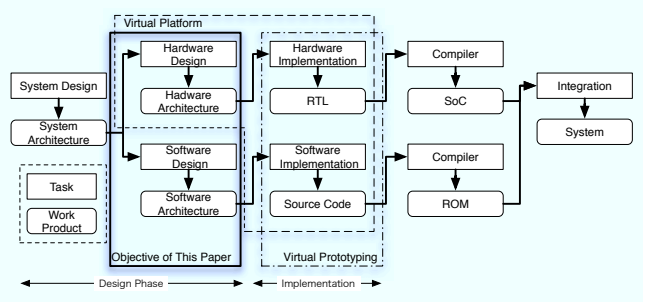


Figure 3. Workflow of Hardware and Software Co-Design with Hardware Virtualization.

to the GW ECU discussed in Sect. II, we can estimate the maximum delay, maximum buffer usage, etc. On the other hand, the simulation-based approach can be adopted for performance analyses. For example, we can estimate several performance parameters such as mean delays and average buffer utilization. Both these approaches are important for system design verification because they can be applied to different design subphases. Although we are actually investigating the analytical approach, we will focus on the simulation-based approach in the remainder of this paper owing to space constraints[1].

Reviewing the discussions in Sect. II, a system design verification technique must be able to handle the following two types of behavior.

1) Inter-ECU behavior: The interconnection of ECUs via networks.
2) Intra-ECU behavior: The interaction between the hardware and the software within ECUs.

In the following sections, we will discuss two different types of approaches that can potentially handle the two behaviors stated above.

A. Network-oriented Approach

The first potential approach to system design verification is to use network simulation technologies. This approach seems to be able to perform end-to-end analysis of automotive systems. Network simulation tools, e.g., OPNET Modeler [7], enable us to simulate the network component, i.e., CAN, LIN buses, etc. Unfortunately, such a simulation is not sufficient for the holistic analysis of complex automotive systems such as the one shown in Fig. 1. As discussed in Sect. II, the GW ECU shown in Fig. 1, which interconnects multiple CAN buses, provides complex functionalities via software. More precisely, it enables priority control as shown in Fig. 2. Thus, it is not feasible to simplify the GW ECU into a simple delay element that can be easily handled by network simulators. In other words, network simulators cannot replicate the detailed behavior of the GW ECU, which is dominant in design verification and must be analyzed through simulation.

[1]See [6] for details of our research on the analytical approach.

B. Hardware Virtualization Approach

The second potential approach is to use co-verification techniques with hardware virtualization that have been established in the ESL (electronic system-level) field. For example, the internal behavior of ECUs is usually analyzed via cycle-accurate techniques featuring instruction set simulators, e.g., CoMET [8]. This type of technique is known as *virtual prototyping*, and its typical workflow is shown in Fig. 3. Virtual prototyping is useful for timeliness verification of systems under development. However, it is clear from Fig. 3 that virtual prototyping can be applied only to test phases because it requires implemented software, i.e., a set of source code. Unfortunately, the cost of eliminating defects in test phases is much higher that in design phases. Furthermore, unlike network simulators, virtual prototyping cannot enable end-to-end analysis of systems while considering network components such as CAN and LIN buses.

Virtualization can also be achieved using *virtual platforms*. Although virtual platforms enable system verification based on a hardware design, software implementation is required. On the other hand, the objective of this paper is to achieve system design verification based on both hardware and software design specifications.

C. Co-simulation Approach

Co-simulation [3] is another potential approach to system design verification. Combining the two techniques described above would be possible from a technical point of view. This approach can enable end-to-end analysis based on the detailed behavior of ECUs. However, the co-simulation approach is not considered pragmatic because it uses multiple tools for the same verification and it involves high running costs.

As discussed in Sects. III-A and III-B, existing techniques are insufficient for the realization of design verification of automotive systems. Thus, a new verification technique is required to deal with the combination of hardware, software, and networks. On the other hand, future automotive systems will be not be standalone systems, and their connectivity will be mandatory. Thus, from the viewpoint that multiple

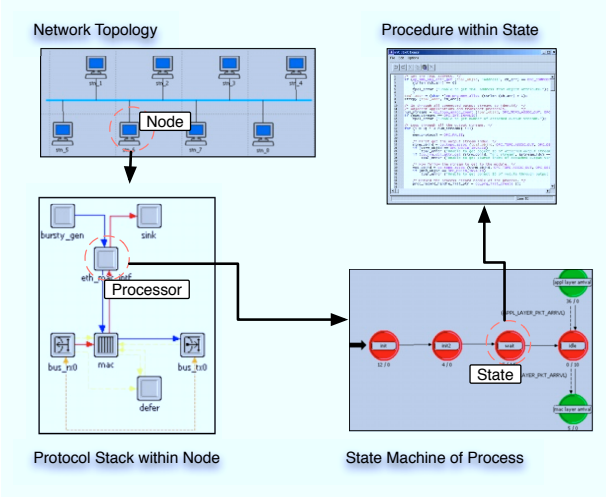


Figure 4. Modeling Steps in OPNET Modeler.

automotive systems in numerous vehicles and a variety of non-automotive systems are interconnected, the network-oriented approach is a more attractive approach owing to its potential in various types of network simulations. Therefore, in this paper, we try to develop a new verification technique by enhancing the *network-oriented* approach.

IV. Holistic Simulation of Automotive Systems

In order to resolve the problems described in Sect. III-A, we propose an abstract behavior model of ECUs, which is obtained by extending the modeling fundamentals of OPNET Modeler [7]. By using the proposed method, we can potentially achieve the holistic simulation of both the network component and the internal component of ECUs (hardware and software).

A. OPNET Modeler

OPNET Modeler enables the modeling and simulation of several types of networks. OPNET Modeler creates network models according to the following four steps, and then realizes network simulation.

1) Network topology consists of multiple *nodes*.
2) Protocol stack within a node comprises *processors*.[2]
3) Behavior of a process is defined by a *state machine*.
4) Procedure within a state is described by C or C++ *source code*.

Thus, OPNET models have a hierarchy that consists of four levels: *nodes*, *processors*, *state machines*, and *code*, as shown in Fig. 4. This modeling approach enables the effective simulation of protocol stacks; e.g., [9] and [10] focus on the modeling of the SIP stack.

[2]The term *processor* is a special keyword in the OPNET Modeler environment. It should be noted that the *processor* of OPNET Modeler is different from well-known processors like x86, ARM, etc.

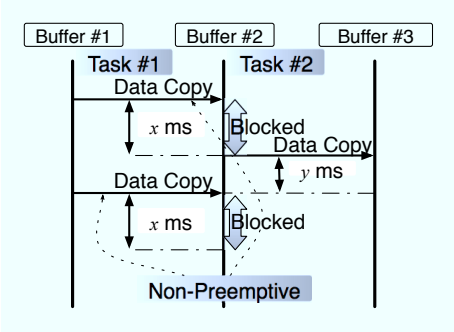


Figure 5. Problems in Behavior Modeling of Software.

OPNET Modeler adopts a simulation scheme whereby the values assigned to states within processors, i.e., delay times, are accumulated. This modeling scheme is effective for the simulation of protocol stacks; however, this scheme cannot simulate the complex behavior of software, which can stop an ongoing task and trigger another task. Therefore, we cannot capture runtime software behavior such as preemption, interrupts, and timeouts by adopting the straightforward scheme described above.

The reason for this problem can be explained by a simple example shown in Fig 5. Figure 5 describes two types of copy operations: the copy from Buffer #1 to Buffer #2 and the copy from Buffer #2 to Buffer #3. These operations are implemented as independent software tasks, Task #1 and Task #2, respectively. Here, let us assume that the priority of Task #2 is higher than that of Task #1; in contrast, the execution time of Task #1 is much longer than that of Task #2, such that $x \gg y$. In this case, it is possible that Task #2 preempts Task #1 in a real system. However, OPNET Modeler cannot simulate this behavior, and every execution of Task #1 finishes in x msec. In the case of OPNET Modeler, Task #1 and #2 are described as independent processors. The procedure of each processor, which is described by the combination of a state machine and source code, is treated as an atomic operation; thus, one processor cannot preempt the other processor. Therefore, OPNET Modeler cannot simulate the preemptive behavior described above. The same problem can occur in the case of software consisting of a single task and some interrupt service routines.

B. Runtime Behavior Modeling of Software

In order to realize a simulation model that captures the internal structure of ECUs, we need a new method to deal with the complex behavior of software by overcoming the problems mentioned in Sect. IV-A. Therefore, we develop the following approach, which is compatible with OPNET Modeler.

1) The software is composed of coarse-grained chunks (hereafter, referred to as *commands*).

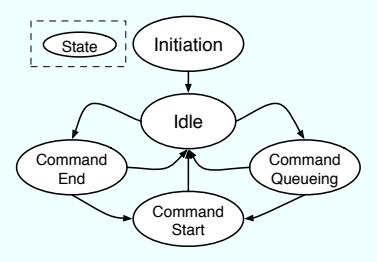


Figure 6. State Machine of Commands.

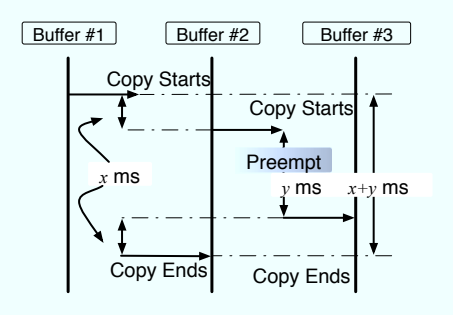


Figure 7. Copy Sequence Model with Preemption.

2) The execution procedure of *commands* is modeled by using a state machine within a processor whose states are Initiation, Queueing, Start, End, and Idle (see Fig.6).
3) The values of delay time are assigned to each *command*.

This modeling approach enables us to execute another *command* between Start and End of one *command*. Moreover, queued *commands* represent low priority software tasks waiting to run. Consequently, preemption, interrupts, and timeouts can be simulated as shown in Fig. 7; hence, we can build a simulation model that can replicate such complicated behavior. In addition, the abstraction level of software composed of *commands* is nearly similar to that of the detailed design of software. Thus, we can regard the command-based decomposition of software as a design specification of software. In other words, we have obtained a simulation technique based on the design specification of software, which can be used to perform the design verification of automotive systems.

The efficiency of the proposed technique is highly dependent on the soundness of *command* models; hence, we must define each *command* with appropriate granularity. When we want to define a part of software with an appropriate size for a single *command*, we can evaluate its validity on the basis of its execution time. On the other hand, it is well known that memory access is a dominant part of most embedded software owing to its slowness. Thus, we can define different types of memory accesses as distinct *commands*, respectively. In the case of Fig. 2, the data copy

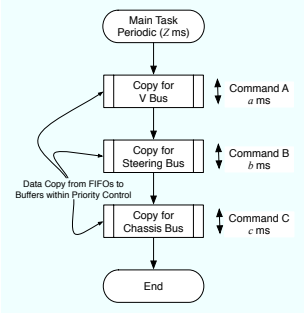


Figure 8. Example Decomposition of Task into Commands

operation occurs from the FIFO for the V1 bus to the corresponding buffer within the following priority control unit as a single *command*; the same copy operation for the steering bus occurs as another command, and so on. The other types of operations unrelated to memory access, e.g., calculation, flow of control, etc., can be treated as a single chunk. For example, Fig. 8 shows the design specification of a periodic task which copies from the FIFOs to the buffers within the priority control units in Fig. 2. As shown in this figure, we can define each copy operation as a single *command*. Although this example is trivial, we can create models for other complex tasks or interruption service routines by following the same modeling strategy. The proposed modeling technique is simple; however, it provides sufficient simulation accuracy (see experimental results in Sec. V).

C. Behavior Modeling of Hardware (Wired Logic) Components

Figure 2 shows that a computation-intensive operation is implemented using hardware. The same design strategy of wired-logic acceleration is common among several ECU designs. One example is the communication stack for FlexRay [11], which requires high computational power; it is executed by a hardware accelerator. Therefore, a simulation technique for design verification must be able to handle these hardware components in its simulation.

Fortunately, we can handle hardware components in an easier manner as compared to software components. This is because the deviation of the latency time of hardware components does not vary much, whereas that of software components dynamically fluctuates along with the runtime scheduling of software tasks. More precisely, most hardware accelerators used in ECUs are composed of combination logic circuits or simple sequential logic circuits, as in the case of Fig. 2. Thus, the latency time of such hardware accelerators can be easily estimated. In addition, the fact that the latency time of hardware components is much shorter than that of software simplifies the simulation models of hardware for delay analysis.

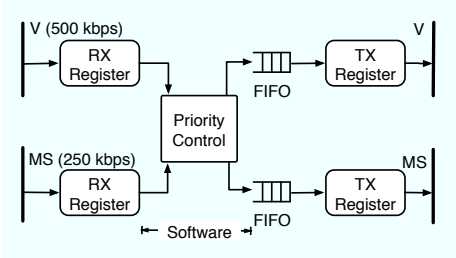


Figure 9. Architecture of Dual-Channel CAN Gateway ECU.

Actually, in the GW ECU architecture in Fig. 2, the hardware component, which executes routing operations, produces constant latency from the RxRegisters to the FIFOs, unless simultaneous receptions of multiple messages from different buses occur. This means that we only have to consider such conflict conditions in simulation. Hence, we can calculate the delay of a certain message from the corresponding RxRegister to FIFO by simply multiplying the latency time of the non-conflict case, i.e., two or three times longer when simultaneously receiving from two or three buses, respectively.

According to the aforementioned modeling strategy, OPNET Modeler can simulate the internal behavior of ECUs while considering both hardware and software. Furthermore, OPNET Modeler is inherently able to simulate CAN buses, which interconnects multiple ECUs, in a straightforward manner. Eventually, combining the three types of models, i.e., the software model, hardware model, and network model, we can perform an exhaustive holistic simulation of automotive systems using a single tool (OPNET Modeler).

V. Experimental Results

In this section, we present some experimental results to verify the effectiveness of the proposed holistic simulation technique. In the experiments, two different types of GW ECUs are used (see Table I). The first gateway (GW ECU #1) is identical to the GW ECU shown in Fig. 2. The second one (GW ECU #2) connects two CAN buses with different speeds, and its architecture is shown in Fig. 9. The simulation results of the proposed technique are compared with those of conventional virtual prototyping using CoMET.

Figures 10(a) and 10(b) show the simulation (OPNET) models obtained by applying the proposed technique to these two GW ECUs. These simulation models enable several types of analyses from the following viewpoints.

(1) Message delay.
- Delay in buffers and GW ECUs.
- End-to-end delay.

(2) Buffer utilization.
- Average buffer utilization.
- Buffer overflow and underflow.
- Maximum buffer usage and minimum buffer usage.

(3) Message lost.

Table I
Specifications of Target Gateway ECUs.

Name	GW ECU #1	GW ECU #2
Type	CAN–CAN	CAN–CAN
Channels	3 channels	2 channels
Buses	V, Steering, and Chassis buses	V and MS buses
Bus Speed	500 kbps	500 kbps (V) and 250 kbps (MS)
Logic	Hardware & Software	Software
Architecture	Figure 2	Figure 9

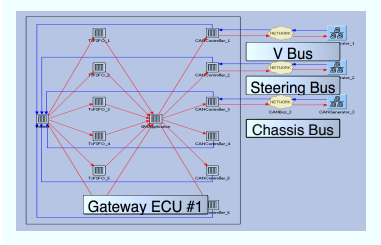
Regarding message delay, comparisons between the proposed OPNET-based simulation and CoMET-based simulation are provided below. Figures 11 and 12 show comparisons of the mean message delays estimated using the proposed technique and CoMET-based simulation. More precisely, Figs. 11(a) and 12(a) illustrate the estimated GW delays that are equivalent to the latency between message reception in RX registers and transmission from TX registers. Figs. 11(b), 11(c), 12(b), and 12(c) partially show the breakdown of the GW delays: estimated delays in RX registers and FIFOs.

In these figures, we can find some deviations of the delays estimated by the two approaches; however, they are approximately less than 20% in most cases. In particular, the proposed technique provides equivalent accuracy in the cases of important messages with high priorities. In addition, comparing Figs. 11 and 12, we can recognize that the proposed technique can achieve high accuracy, irrespective of the architectures of simulation targets (see Figs. 2 and 9).

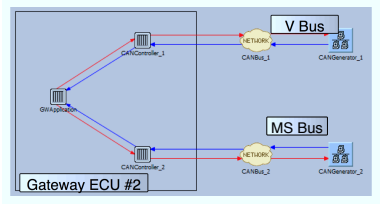
Figures 13(a) and 13(b) show comparisons of estimated end-to-end delays, which is the latency time between message transmission from source ECUs and reception in sink ECUs. This end-to-end delay analysis was infeasible when we used only CoMET-based simulation; however, they were feasible using the proposed holistic simulation technique.

From the viewpoints of buffer utilization such as overflow, underflow, and maximum usage, and message lost, the results obtained by using both the simulations are identical. This implies that the deviations of the simulation results shown in Figs. 11 and 12 do not have any impact on the architecture decisions of the GW ECUs.

Based on the discussion presented above, the proposed technique provides sufficient accuracy despite its coarse granularity. Moreover, the proposed technique enables an end-to-end delay analysis, unlike the CoMET-based technique. Besides the accuracy, the simulation speed of the proposed method also enhances its effectiveness. The proposed

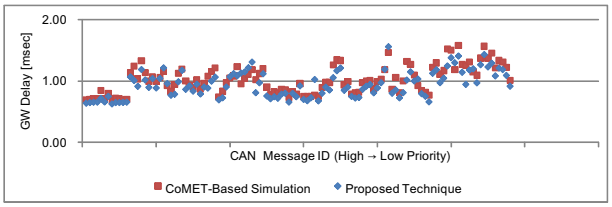


(a) GW ECU #1 and CAN buses.

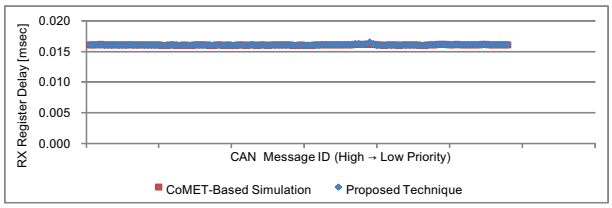


(b) GW ECU #2 ECU and CAN buses.

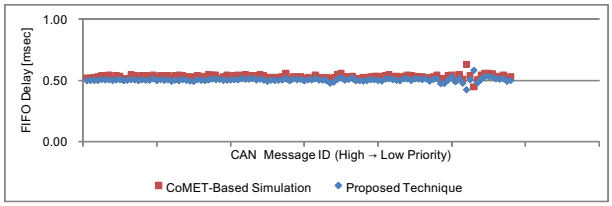
Figure 10. OPNET Models of Distributed Automotive Systems.



(a) Mean GW Delay.

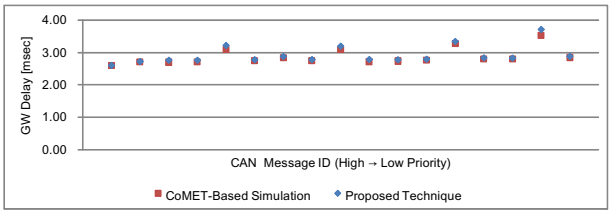


(b) Mean Delay in Rx Register

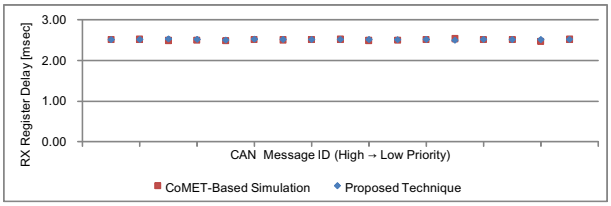


(c) Mean Delay in FIFO.

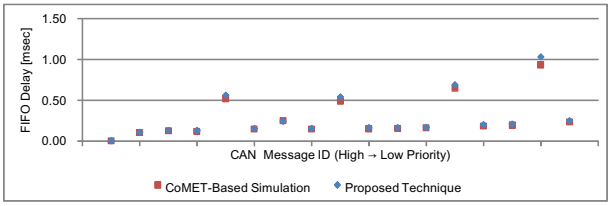
Figure 11. Comparisons between Simulation Results of OPNET and CoMET Models (GW ECU #1).



(a) Mean GW Delay.

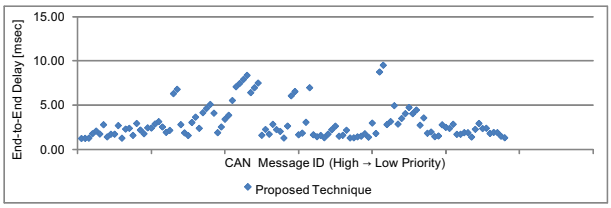


(b) Mean Delay in Rx Register

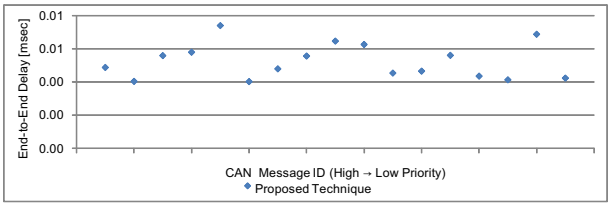


(c) Mean Delay in FIFO.

Figure 12. Comparisons between Simulation Results of OPNET and CoMET Models (GW ECU #2).



(a) Simulation Results of GW ECU #1.



(b) Simulation Results of GW ECU #2.

Figure 13. End-to-End Delay Estimated by Proposed Technique.

simulation technique is around 1.5 times faster than execution in a real-time environment, even under an ordinary PC environment (Intel Core 2 Duo 2.33GHz); such acceleration cannot be expected in the case of fine-grained simulation such as virtual prototyping.

Summarizing the quantitative discussion presented above on the basis of the experimental results and the qualitative discussion presented in Sects. III and IV, we derive a comparison between the proposed technique and CoMET-based simulation, as shown in Table II. The table shows that each approach has different characteristics, and hence, different applicable phases. This implies that we can use both approaches in a collaborative way for different types of verification, i.e., the proposed technique is used for design verification, and CoMET-based simulation is used for implementation verification.

Table II
COMPARISON BETWEEN PROPOSED AND COMET-BASED SIMULATION.

Approaches	Proposed	CoMET-Based
Coverage	Software, hardware, and network	Software and hardware
Abstraction	High	Low
	Design models (e.g. Fig. 8)	Source code (software and hardware)
Accuracy	Medium	High
Simulation Speed	High	Medium
Applicable Phase	Design	Test
Required Literacy	C language	C language and HDL

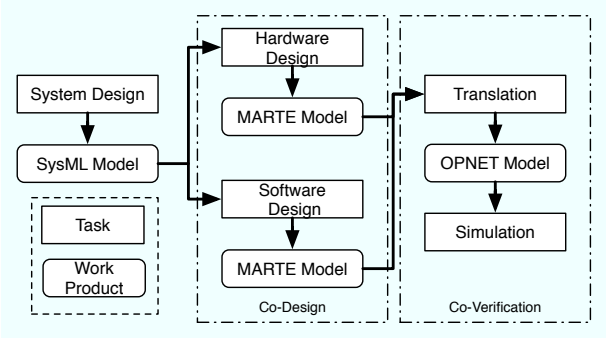


Figure 14. Model-Based Development using Proposed OPNET-based Simulation (Future Work).

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed an OPNET-based simulation of real-time distributed automotive systems. The proposed technique can deal with the three important components of automotive systems, i.e., hardware and software within ECUs, and networks among ECUs. Thus, it enables a holistic simulation of automotive systems. Moreover, the end-to-end analysis is a new feature enabled by the proposed holistic simulation, which was infeasible when we used CoMET-based simulation. The proposed technique enables design verification; hence, the efficiency of system development can be potentially enhanced by using this technique. In addition, the proposed technique depends on a single commercial tool; hence, it has low running costs and it can be immediately deployed for real-world development.

In the future, it will be necessary to ensure that the proposed technique conforms with standardized modeling methods, e.g., SysML, MARTE, etc. Once the alignment with these standards is established, model-based development in close collaboration with the proposed simulation technique should be possible, as shown in Fig.14.

VII. ACKNOWLEDGEMENT

The author would like to extend his gratitude to Fumiyuki KAGARA (Toyota Technical Development), Hiroki KONDO and Naoki OHTA (CIJ) for their contribution.

REFERENCES

[1] *ISO/DIS 26262 Road vehicles – Functional safety –*. ISO/TC22/SC3/WG16, June 2009.

[2] C. Lauer, R. German, and J. Pollmer, "Discrete event simulation and analysis of timing problems in automotive embedded systems," in *Systems Conference, 2010 4th Annual IEEE*, 2010, pp. 18 –22.

[3] M. Karner, E. Armengaud, C. Steger, and R. Weiss, "Holistic simulation of flexray networks by using run-time model switching," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 544 –549.

[4] *CAN Specification Version 2.0*. R. Bosch GmbH, September 1991.

[5] *LIN Specification Package Revision 2.0*. LIN Consortium, September 2003.

[6] M. Sebastian, P. Axer, R. Ernst, N. Feiertag, and M. Jersak, "Efficient reliability and safety analysis for mixed-criticality embedded systems," in *the SAE World Congress*, 2011, pp. 544 –549, to be published.

[7] *OPNET Modeler*. http://www.opnet.com.

[8] *CoMET*. http://www.synopsys.com.

[9] H. Atrianfar and Z. Ayatollahi, "Expansion of OPNET modeler's SIP model for performance evaluation of hierarchical call routing in NGN," in *International Conference on Scientific Computing (ICSC 2008)*, 2008, pp. 179 –185.

[10] T. Le, S. Cook, G. Kuthethoor, P. Sesha, G. Hadynski, D. Kiwior, and D. Parker, "Performance analysis for SIP based VoIP services over airborne tactical networks," in *Aerospace Conference, 2010 IEEE*, 6-13 2010, pp. 1 –8.

[11] *FlexRay Communications System Protocol Specification Version 2.1*. FlexRay Consortium, May 2005.

Notes

Notes

A Simulation Framework for Design of Mixed Time/Event-Triggered Distributed Control Systems with SystemC/TLM

Zhenkai Zhang, Joseph Porter, Xenofon Koutsoukos, and Janos Sztipanovits
Institute for Software Integrated Systems (ISIS)
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, Tennessee, USA
Email: {zhenkai.zhang, joseph.porter, xenofon.koutsoukos, janos.sztipanovits}@vanderbilt.edu

***Abstract*—Mixed time/event-triggered (TT/ET) distributed control systems are complex systems which have emerged in many cyber-physical domains but have been difficult to evaluate at early design stages. In order to reveal design flaws as early as possible, this paper proposes a simulation framework based on an executable virtual platform model in SystemC/TLM. The executable platform is generated using a model-based approach from a system designed in the Embedded Systems Modeling Language (ESMoL). The virtual platform consists of three types of abstract models, the RTOS model, the communication system model, and the hardware model, to capture different behaviors of the mixed TT/ET distributed control systems. Preliminary results from a case study using a Quadrotor flight control system are used to illustrate the approach.**

***Keywords*-Mixed Time/Event-Triggered Distributed Control Systems; Virtual Platform; SystemC/TLM; Graphical Models;**

I. Introduction

Nowadays, most complex cyber-physical systems (CPSs), such as automotive vehicles, air planes and trains, use distributed control systems, in which several ECUs are connected by network(s)/bus(es) [22]. Typically, these control systems are hard real-time systems. Traditionally, these control systems are composed of event-triggered (ET) tasks and use event-triggered communication systems, such as CAN bus. As time-triggered architectures (TTA) offer advantages such as determinism, predictability, and composability [9], many new systems tend to be built using TTA. However, many sporadic events make the designs not fit into the strict periodic framework [13]. These CPS control systems often consist of both TT and ET tasks and use a mixed communication protocol (e.g. FlexRay and TTEthernet) to form mixed TT/ET distributed control systems [17].

When designing mixed TT/ET distributed control systems, many challenges arise due to a large design space and lack of tools to explore it. First, the hardware platform needs to be designed including a set of nodes connected by a communication system. Trade-offs between cost and performance drive the selection of the appropriate processors. The communication system bandwidth and topology also need to be considered in order to meet performance and redundancy requirements. Then, partitioning tasks into TT or ET needs to be considered. Moreover, mapping the tasks on the hardware platform is also important and affects timing [17]. After mapping, the mixed communication system needs to be configured using proper parameters. Thus, the space of possible design configurations is large and early evaluation is necessary to eliminate bad design decisions which may cause the system to fail to meet its requirements at a later design stage. However, most existing work focuses on specifying the control system [23], analyzing the system's schedulability [18], optimizing partitioning, mapping and bus cycle [17], and inter-task communication mechanism [21], but not evaluating the whole system.

For mixed TT/ET distributed control systems, both computation and communication should be captured to enable evaluation of the whole system with respect to functionality, timing, and performance. Both computation and communication concerns are coupled to the application software, system software and hardware platform. The ability to model, integrate, and simulate all parts together is essential for design space exploration during early development. A virtual platform including both hardware and embedded software can be used as a pivot in this evaluation framework, since it is available much earlier than the real system [7].

System-Level Design Languages (SLDLs), such as SystemC and SpecC, can be used to model both hardware platforms and embedded software. In addition, most SLDLs support the concept of transaction-level modeling (TLM) which separates the design of the computation and communication. A TLM communication structure abstracts away communication details to speed up simulation while keeping required accuracy. SystemC has been a *de facto* SLDL [3]. It also has a TLM library for modeling memory-mapped buses. SystemC/TLM-based virtual platforms on system-level can model the hardware behavior with good simulation efficiency and sufficient timing accuracy at early stages [4].

In order to evaluate a mixed TT/ET system, we propose a simulation framework based on a virtual platform in SystemC/TLM and is combined with a model-based design

environment in GME [12]. The virtual platform consists of a mixed TT/ET computation model and communication model. For the computation model, an abstract RTOS model and abstract hardware models such as processor and peripherals are needed. The abstract RTOS model is built in SystemC and takes charge of task scheduling, inter-task communication, synchronization and interrupt handling [5] [11] [24]. The behaviors of both TT and ET tasks should be captured by the abstract RTOS model, and abstract hardware models form the underlying computational platform. For the communication model, an abstract communication system model which can capture the behaviors of both TT and ET communication is needed. Since FlexRay has been widely used in many CPS domains, it is a good reference for modeling the mixed TT/ET communication [2]. The abstract communication system model in this framework is based on FlexRay.

The simulation framework is also integrated with a model-based design environment in GME called Embedded Systems Modeling Language (ESMoL) [19]. The mixed TT/ET distributed control systems in ESMoL can be transformed to the virtual platform models using automated model transformations. This integration makes the design and simulation of the mixed TT/ET systems effective and efficient. As the framework is still in progress, preliminary results from a case study using a Quadrotor flight control system are used to illustrate the approach. The bandwidth of two communication systems is evaluated for the designed mixed TT/ET system.

There have been efforts to establish simulation frameworks for design of distributed real-time control systems. In [15], a framework based on generated virtual execution platform is proposed, in which VaST tool is used to model a cycle-accurate hardware platform and μC/OS-II RTOS is ported to the modeled platform. Although the simulation can be very accurate, the very low abstraction level makes it not suitable for early design evaluation. Compared to [15], a framework named E-TTM is proposed on a very high abstraction level for the design of TTA-based real-time control systems in [16]. Too high level of abstraction also impedes the use of the framework due to inaccuracy. In [14], a UML-based design framework is proposed. A system is described in UML, and then the UML model can be converted into SystemC model for simulation. However, this framework is only for TT applications without considering mixed TT/ET systems.

Compared to the related efforts, the main contributions of this proposed framework are: (1) it uses abstract computation and communication models to establish a universal simulation framework for mixed TT/ET systems, while the levels of abstraction are appropriate to keep the simulation efficient and accurate; (2) the framework is also integrated with a model-based design tool to improve its usability.

The rest of this paper is organized as follows: Section 2 introduces the virtual platform model in the proposed simulation framework which consists of three types of models; Section 3 describes how to transform an ESMoL design into an executable virtual platform; Section 4 gives a case study on the design of a Quadrotor flight control system and uses the simulation framework to evaluate the influence of the bandwidths of the communication systems; Section 5 concludes this paper and gives future work.

II. Virtual Platform Model

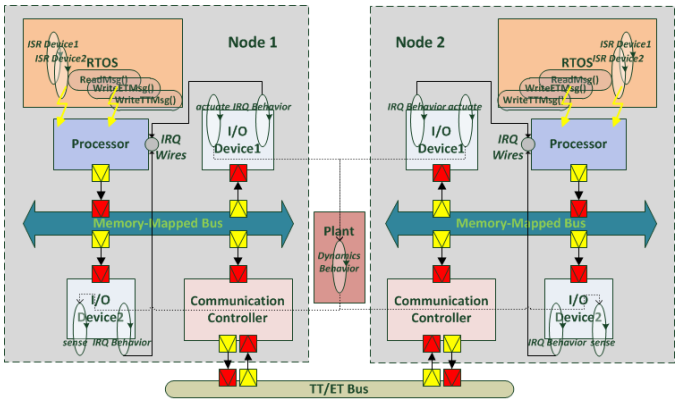


Fig. 1. Virtual platform with abstract RTOS model, abstract communication system model, and abstract hardware model.

The virtual platform, as shown in Fig. 1, consists of three types of models which are the abstract RTOS model, the abstract hardware model, and the abstract communication system model. The models are implemented in SystemC by inheriting from the *sc_module* class in which concurrent behaviors are modeled by a set of SystemC processes (*SC_THREAD* or *SC_METHOD*). These three models can be instantiated and integrated to be an executable model for simulating mixed TT/ET control systems.

A. RTOS Modeling

On a node of a distributed system, TT and ET tasks which realize the desired functionalities interact with an RTOS. The RTOS captures the dynamic behaviors of the tasks.

The abstract RTOS model has three *SC_THREAD* processes, which are the RTOS service process interacting with TT and ET tasks, the time-trigger process taking charge of triggering TT tasks according to a static schedule, and the interrupt handling process invoking an interrupt service routine (ISR) to handle the corresponding interrupt. Fig. 2 shows the fundamental services supported by the abstract RTOS model.

1) Task Management: In a mixed TT/ET system, tasks are divided into TT tasks and ET tasks. Time-triggered tasks are activated according to a predefined schedule. When a node's synchronized local clock reaches a predefined time instant, the corresponding TT task will be put into the ready queue. TT tasks can be non-preemptive or preemptive. ET tasks are activated dynamically depending on the occurrence of associated events. ET tasks can also be non-preemptive or preemptive.

Each task in the abstract RTOS model corresponds to a SystemC *SC_THREAD* process. In order to serialize the tasks and control their execution, each task pends on its own *sc_event* object. The RTOS scheduler controls the execution by notifying the task's *sc_event* object. A task's execution

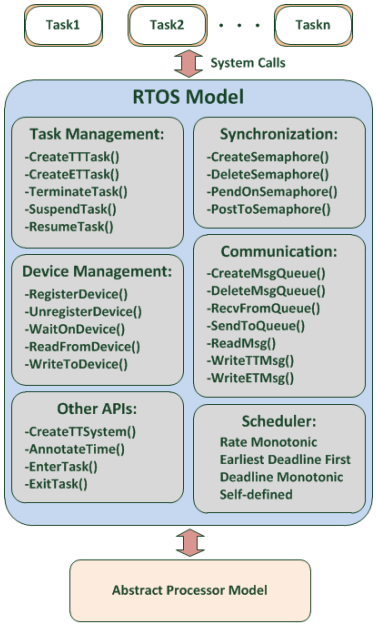


Fig. 2. Abstract RTOS model with supported primitives

information is stored in its Task Control Block (TCB). Since in SystemC the execution between two *wait()* statements is in zero simulation time, we need to advance time and model execution of tasks by using *wait()* statements. The execution time of a section of code is modeled by inserting timing annotations into the task. The annotation can be coarse-grained on the task level or fined-grained on the basic block or statement level.

There are a set of primitives provided by the RTOS model to manage a task's creation, termination, resumption and suspension. When created, each task needs a task name, a worst case execution time (WCET), and a deadline. In addition, each TT task also needs a predefined schedule passed as a parameter, and each event-triggered task needs a user-defined priority and/or a period depending on the scheduling policy.

2) Scheduling: The scheduler is the heart of the RTOS, which allocates CPU time to a selected task from the ready queue. The scheduler's behavior depends on a specific scheduling algorithm. In the abstract RTOS model, the scheduler has three common priority-based scheduling policies which are rate monotonic (RM), deadline monotonic (DM), or earliest deadline first (EDF). Other scheduling algorithms can also be easily added into the RTOS model. The scheduler's timing properties are RTOS- and hardware platform-specific. Basically, there are two main parameters of its timing properties: scheduling overhead and context switching overhead. There is some research work on how to accurately acquire these parameters [6] [8], which is not the focus of this paper, so we assume the parameters are already available for a particular system.

The task state transitions are modeled by two finite state machines as shown in Fig. 3, one for TT tasks and the other one for ET tasks. The *Created* state is for any new task. Depending on the task type (TT or ET), it transitions to the corresponding state. A TT task enters the *Idle* state, and an ET task enters the *Ready* state. A TT task enters the *Ready* state statically according to an *a priori* schedule table, while an ET task enters the *Ready* state dynamically when the event happens. There is only one ready queue, which contains both time-triggered and event-triggered "ready-to-run" tasks. The scheduler schedules this ready queue using the assigned scheduling policy. Only one task can be in the *Running* state at a time which is chosen by the scheduler.

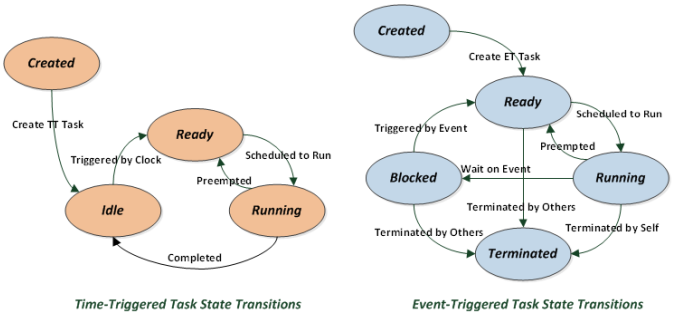


Fig. 3. Task state transitions of TT/ET tasks.

3) Inter-Task Communication: In a multi-tasking RTOS, tasks need to communicate with others synchronously or asynchronously using inter-task communication mechanisms. In the abstract RTOS model, inter-task communication on one node can be achieved by shared memory or message queue, and inter-task communication between different nodes can be achieved by message-passing. Shared memory is used between TT tasks, since it can be accessed without race-conditions. Semaphore synchronization is used by ET tasks to serialize access to shared memory and maintain task dependencies.

Communication between TT and ET tasks on a single node occurs through message queue. An agent is associated with a TT task in the message queue as a state message keeper. If the message queue is empty, when an ET task tries to poll a message from it, it will be blocked on it; whereas, for a TT task, the agent will give the task the message in the queue if it is available, and then update its state message as the latest dequeued message or give the task the state message if the queue is empty. So when a TT task accesses the message queue, it will never be blocked even if there is no message available in the queue.

Communication between two tasks on different nodes are through message passing. Two types of messages are supported in the model, one is TT and the other one is ET. TT messages are transmitted in the predefined static time slots of a communication cycle. ET messages are transmitted according to the combination of their priorities and the dynamic slots.

4) Interrupt Handling: SystemC has some disadvantages for RTOS modeling, which can be summarized as non-interruptible *wait-for-delay* time advance and non-preemptive simulation processes. When an interrupt happens, it requires the real-time system to react and handle it in a timely manner. Modeling an accurate interrupt handling mechanism plays an important role in RTOS modeling. We adopt the method from

[24] which makes task use *wait-for-event* other than *wait-for-delay* to advance its execution time. A system call of the RTOS model taking execution time as its argument makes the task wait on a *sc_event* object which will be notified after the given execution time elapses if no interrupt happens. When an interrupt happens and its corresponding ISR preempts the execution of the task, the notification of the *sc_event* object will be canceled and a new notification time will be calculated according to how much time the preemption took and how much execution time already passed.

B. Communication System Modeling

In a distributed control system, the timing behavior of the communication system has an important impact on system performance. There are a few communication protocols that provide predictable message delays [20]. The abstract communication system model in this framework is based on FlexRay protocol [2], which can handle both time-triggered and event-triggered communication. The data granularity of the model is at the message-level, since we only need to consider message delay rather than the detailed timing of underlying operations for evaluation of the system. The behavior of the communication system is modeled by a state chart as shown in Fig. 4. The abstract communication system model takes advantage of the global time in the SystemC simulation kernel, and uses it as its synchronized time base.

The communication controller model realizes the behavioral model of the communication system and takes charge of transmitting and receiving messages through the underlying medium. Its implementation class is also derived from the *sc_module* class of SystemC. It also utilizes the TLM-2.0 library in SystemC to realize the underlying transmission. The TLM-2.0 library is mainly for modeling memory-mapped buses, so we change some of its semantics to model our communication system. The controller acts as both an initiator and a target for TT/ET bus transactions, and the TT/ET bus is an interconnect component. The controller also acts as a target for memory-mapped bus transactions within a node. The write command of a transaction means to transmit the message included in the generic payload. For simplicity, only the blocking transport interface (*b_transport()* method) is used.

Bus communication is organized in cycles. Each cycle consists of three segments including a time-triggered static segment, an event-triggered dynamic segment and a waiting segment. The time-triggered part is based on a time-division multiple-access (TDMA) medium access protocol (MAC), and the event-triggered part is based on flexible TDMA as in FlexRay and Byteflight [2] [1]. For time-triggered communication, a predefined schedule is also needed and passed through a configuration file.

In the static segment superstate, if the current time slot is scheduled to receive a message from the bus, the communication controller goes into *RECV* state; if scheduled to send a message, it transitions to the *SEND* state to start a message transmission. Otherwise, it will stay in the *IDLE* state. When a static time slot is elapsed, the model checks whether the time-triggered communication part is finished by comparing if the slot counter has reached the allocated number of static slots. The schedule guarantees there are no transmission contentions, so dynamic arbitration is not necessary.

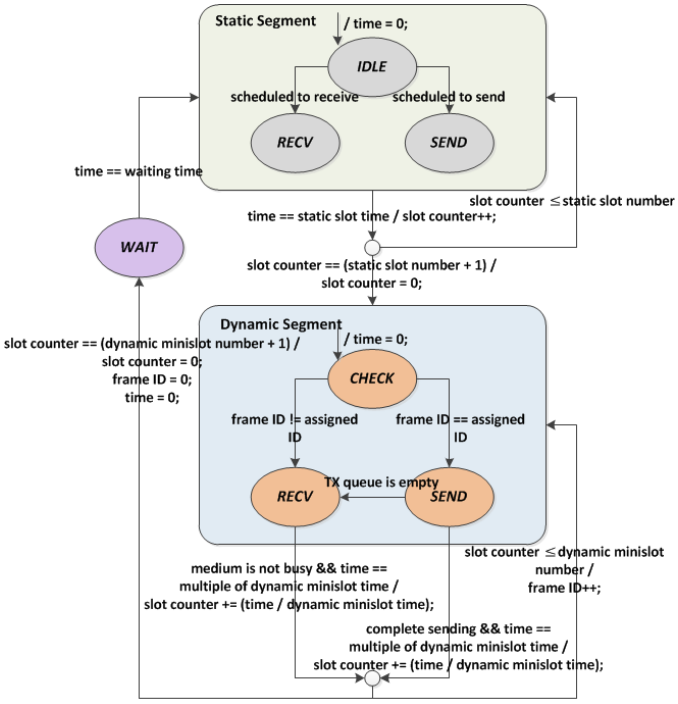


Fig. 4. Behavior of the abstract communication system model based on FlexRay protocol.

The dynamic segment superstate takes charge of event-triggered communication. The segment is divided into a set of minislots. A frame ID variable is updated synchronously by every node in the system. In order to solve the contentions between sending nodes, the rights of transmission are ordered by the frame ID assigned to each node. Different from FlexRay, in this model the dynamic messages are put in a single queue. The queue is sorted by the priorities assigned to the messages associated with the task's priorities, and the messages with the same priority are ordered by FIFO. Each dynamic time slot can have varying number of minislots. The length of a dynamic time slot depends on the size of the transmitted message. When the controller transmits the data in its queue, the controller needs to check whether this is allowed. First, it checks if it has the right to use the current frame ID by comparing with its assigned frame IDs. Then, it searches in the queue for the message with the highest priority which can fit into the remaining dynamic segment time. If there is such a message, the controller sends it on the bus by calling *b_transport()* method; otherwise, the controller defers sending and will try in the next cycle. When the dynamic communication phase is finished, the controller enters the *WAIT* state. In FlexRay, this time is mainly for clock synchronization. Since we use the global time in the SystemC simulation kernel, we do not need to do clock synchronization and we use this state to model a realistic timing behavior.

C. Hardware Modeling

The abstract RTOS model is running on an abstract processor model which communicates with other peripherals through a memory-mapped bus modeled in TLM-2.0, as shown in Fig. 1. Peripherals are divided into communication controllers and other I/O devices. I/O devices are used to model sensors and actuators that interact with the plant dynamics. Each I/O device has a corresponding ISR *SC_THREAD* process in the RTOS registered when calling *RegisterDevice()*. In the current state of the framework, the processor is modeled in a simplified way. The abstract RTOS model interacts with the processor model by: (1) the tasks invoke *ReadMsg()*, *WriteTTMsg()*, and *WriteETMsg()* primitives to make the processor initiate bus transactions with the communication controller; (2) the processor signals the RTOS model that a registered I/O device's ISR needs to be activated; (3) the ISR signals the processor to start bus transactions with the corresponding I/O devices. The processor model has a *sc_port* object which is a multi-port connected by each I/O device's interrupt request (IRQ) wire.

Each I/O device is derived from the *sc_module* class and has a *SC_THREAD* process to control its IRQ behavior. The behavior of the IRQ can be modeled in two modes, asynchronously periodic and sporadic. For example, an UART's IRQ can be modeled as sporadic if the interval between two interrupts has a minimum period, or it can be modeled as asynchronously periodic if the interrupts have a fixed period but are not synchronized with the clock of the processor. When an IRQ occurs, the I/O device will trigger the IRQ wire connected to the processor and corresponding ISR will become ready to handle the IRQ. The ISR will be put into the ready queue first usually with the highest priority, then it would preempt other tasks and run immediately. The order of interrupt handling is based on the IRQ priorities if there are more than one IRQs at the same time. When an ISR finishes, it will check if there is any ET task pending on it and put the corresponding blocked task into the ready queue. Each I/O device also has a *SC_THREAD* process to interact with the plant model. If the device is a sensor, a *sense* process will pull sensor data from the plant and wait for a read transaction. If the device is an actuator, an *actuate* process will wait for a write transaction and send the data to the plant.

III. Model-Based Approach

The front end of this simulation framework is a single multi-aspect embedded software design environment called Embedded Systems Modeling Language (ESMoL) [19]. The executable simulation model is generated from ESMoL. The model transformation process is shown in Fig. 5. Two interpreters are used to realize the model transformations.

An ESMoL model consists of different models used to capture different aspects of the designed system. The design entry of an ESMoL model is to specify the control system's functionality in the Simulink environment. The Simulink model will be imported into the ESMoL automatically to become the functional specification for instances of software components. A logical software architecture model is established to capture data dependencies between software component instances independent of their distribution over different processors. A hardware platform model is defined hierarchically as hardware units with ports for interconnections. By mapping software components to processing nodes and data messages to communication ports, a deployment model is created. By attaching timing parameter blocks to components and messages, a timing model is established. The whole design process is described in detail in [19].

The interpreter in stage 1 transforms the ESMoL model to an equivalent model in an intermediate language called ESMoL_Abstract. The model in this intermediate language is flattened and the relationships implied by structures in ESMoL are represented by explicit relation objects in ESMoL_Abstract. This translation is similar to the way a compiler translates concrete syntax first to an abstract tree, and then to intermediate semantic representations suitable for optimization. The interpreter in stage 2 uses the UDM model navigation API to generate the simulation model according to the corresponding templates. The generation of the simulation model consists of three parts.

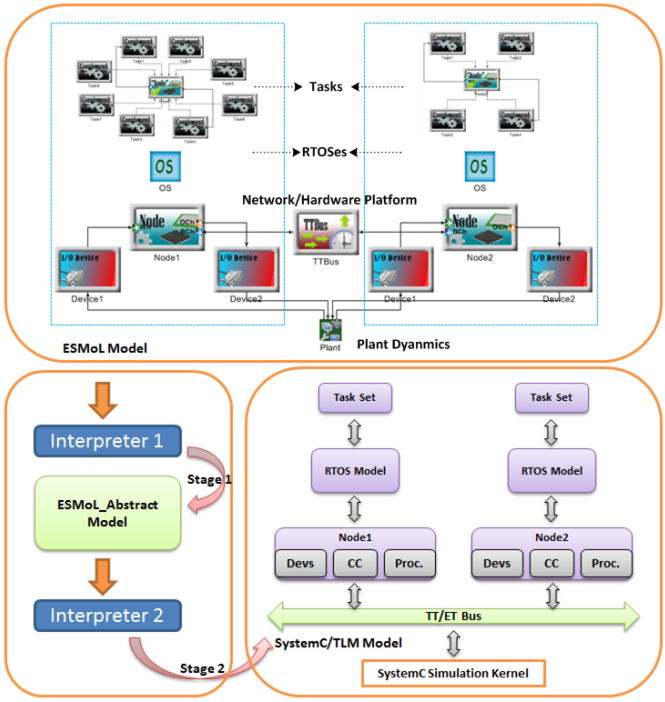


Fig. 5. ESMoL model and its corresponding SystemC model via model transformation using two interpreters.

The first part is to instantiate the hardware and software models according to the templates. Each processor, I/O device, communication controller, bus, and RTOS in the ESMoL_Abstract model is instantiated in the *sc_main()* function. All the instances belonging to the same node are assembled by binding the sockets or ports. The communication controller in each node is bound to the TT/ET bus instance. Each I/O device is registered into the RTOS by calling *RegisterDevice()*

method, which will register an ISR *SC_THREAD* process with its timing and type information (sensor/actuator) passed through the ESMoL model. ISR process pends on its own *sc_event* object, and has the address information of the device. Each task has a corresponding *SC_THREAD* process. The process pends on its own *sc_event* object which will be notified if the process is chosen to run by the scheduler. A task is time-triggered if its *ExecInfo* object in the ESMoL model is a *TTExecInfo* object, whereas it is event-triggered if its *ExecInfo* object is a *AsyncPeriodicExecInfo* or a *SporadicExecInfo* object. A TT task only waits on its own *sc_event* object. An ET task waits on the corresponding event by calling either *WaitOnDevice*, *WaitOnSemaphore* or *WaitOnMsgQueue* primitive. When the event occurs, the ET task goes to the ready queue. All the tasks are registered into the RTOS instances by calling either *CreateTTTask()* (time-triggered) or *CreateETTask()* (event-triggered) primitive. If the task is a sender of a message, it invokes *WriteTTMsg()* system call to send a TT message, or *WriteETMsg()* if the message is an ET one. Shared variables are used for inter-task communication of the same task type (TT/ET). Communication channel between a TT task and an ET task on the same node in the ESMoL model is translated to a message queue. The plant model also has a *SC_THREAD* process for time stepping its dynamics function and is instantiated in the *sc_main()* function. This process also exchanges data with sensors/actuators via shared memory.

The second part generates the configuration files for the model instances according to the specified attributes in the ESMoL model, such as the static schedule tables for RTOSes and the segment configurations for communication controllers. The third part is to generate the functional C code for the tasks and the plant dynamics using Real-Time Workshop (RTW), and integrate the functional code with the generated model in the first part. The generated codes can be easily wrapped into the corresponding *SC_THREAD* processes of the tasks and the plant model.

IV. Case Study

In this section, we employ the simulation framework for the design of a Quadrotor flight control system and present some preliminary results to illustrate the approach.

The controller for the Quadrotor is designed using two linear proportional derivative (PD) controllers, an inner loop and an outer loop, as shown in Fig. 6. The outer loop controller is

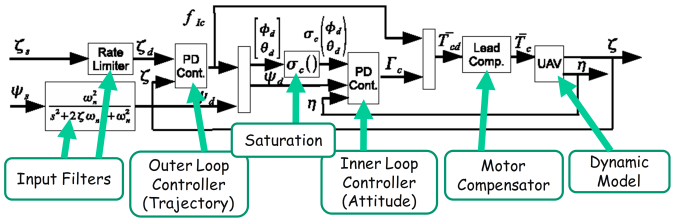


Fig. 6. Two PD controllers in the Quadrotor control system.

a "slow" PD inertial controller and the inner loop is a "fast" PD attitude controller. More details are described in [10]. The corresponding Simulink model (shown in Fig. 7) is built which has four blocks (*ReferenceHandler*, *DataHandler*, *InnerLoop* and *OuterLoop*). After validation of the Simulink model, the model is automatically imported into ESMoL.

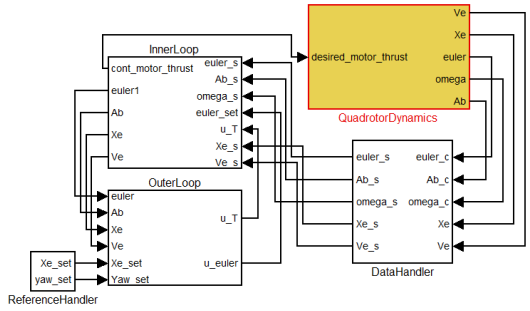


Fig. 7. Simulink model of the Quadrotor control system.

Four ESMoL models are established to capture different aspects of the design. As shown in Fig. 8, the logical software architecture in the ESMoL model gives the data dependencies of the tasks in the Quadrotor control system. Fig. 9 shows the platform of the Quadrotor which has two nodes, one is based on PXA255 processor and the other one is based on ATmega128, and they are connected by a TT/ET bus. Each node has its I/O devices to interact with the outside plant. For task deployment as shown in Fig. 10, two tasks (*ReferenceHandler* and *OuterLoop*) are assigned to the PXA255 node, and the other two tasks (*DataHandler* and *InnerLoop*) are assigned to the ATmega128 node. A timing model (Fig. 11) is also established by attaching timing parameter blocks to components and messages. As shown in Fig. 11, the *ReferenceHandler* task and *DataHandler* task are assigned as event-triggered tasks, and the other two tasks are time-triggered tasks.

In this design, we constrain the design space by using the above hardware platform, task type (TT/ET) assignment, and task deployment, and only focus on the influence of the bandwidth of the TT/ET bus.

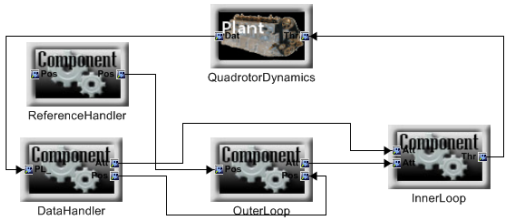


Fig. 8. Software data dependencies of the Quadrotor control system.

The sampling period of the system is 20ms. For tasks on each node, their WCETs are measured empirically. On the PXA255 processor, *ReferenceHandler* needs 50μs with relative deadline 5ms, and *OuterLoop* needs 1.6ms with relative deadline 2ms. On the ATmega128 processor, *DataHandler* needs 200μs with relative deadline 4ms, and *InnerLoop* needs 600μs with relative deadline 1ms. The ISR of the Ethernet on

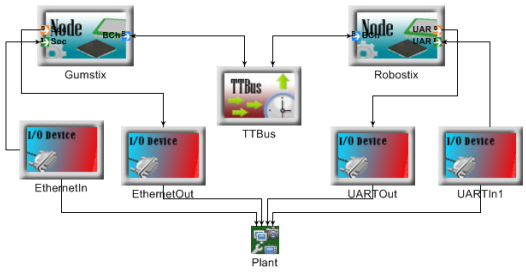


Fig. 9. Hardware platform of the Quadrotor control system.

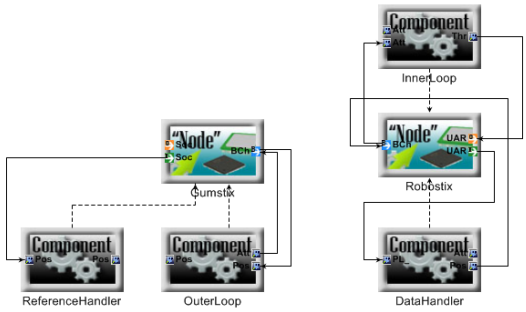


Fig. 10. Task deployment of the Quadrotor control system.

PXA255 needs 5μs, and the ISR of the UART on ATmega128 needs 2μs. In each cycle, *OuterLoop* is time-triggered at 12ms and sends a message to *InnerLoop* with the attitude control data, *InnerLoop* is time-triggered at 10ms, and *DataHandler* sends a messge to *OuterLoop* with the position data.

Since there are a few tasks contending for computation resources, the scheduling algorithm will not make a big difference for control performance. On PXA255 processor RM is used and on ATmega128 processor EDF is used. However, the Quadrotor control system is sensitive to communication delays, which will make us choose the appropriate communication system. Suppose there are only two options for the communication system, one has the maximum bandwidth of 400Kbit/s but cheaper and the other has the maximum bandwidth of 2Mbit/s but more expensive.

First, the 400Kbit/s bandwidth TT/ET bus is tried and the static time slot is set as 2ms, since the largest message (60 bytes) needs 1.2ms for transmission. PXA255 uses the seventh static slot to transmit the attitude control data and ATmega128 uses the sixth static slot to transmit the position data. The simulated control performance is shown in Fig. 12. The left figure shows the simulated trajectory of the Quadrotor compared to its reference, in which the solid lines give the trajectory and the dotted lines give the reference: red lines are positions along X-axis, black lines are positions along Y-axis, and blue lines are height positions, respectively. The right figure shows the error between the simulated trajectory and reference. The timing behaviors of the tasks is shown in Fig. 13. In the timing diagram, each red dotted line represents the deadline of the task. From the timing diagram we can see every task meets its deadline. However, the control performance begins more and more unstable as time passes.

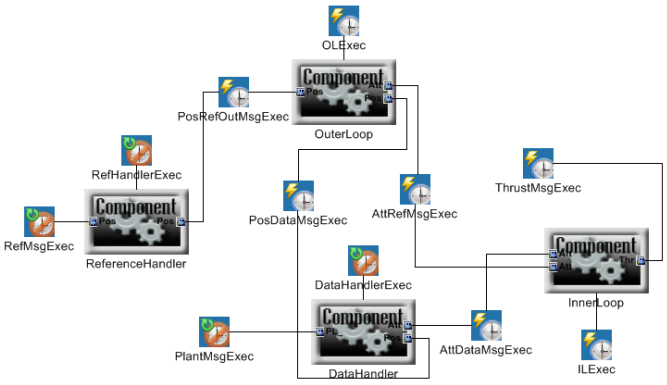


Fig. 11. Timing model of the Quadrotor control system.

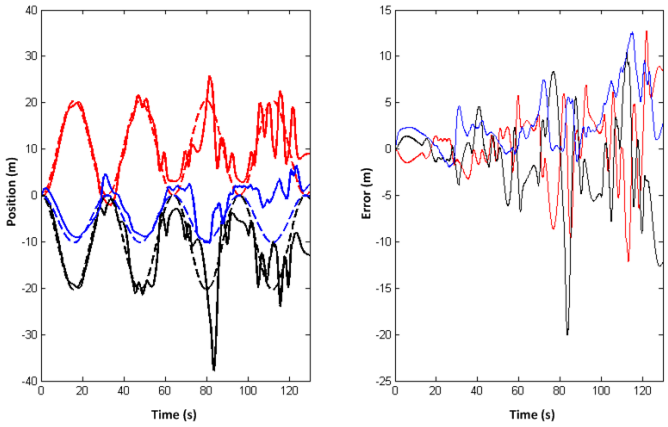


Fig. 12. Control performance with bus bandwidth of 400Kbit/s.

The second case uses the 2Mbit/s bandwidth TT/ET bus and the static slot is set as 1ms, as the largest message (60 bytes) needs 240μs for transmission. PXA255 uses the fourteenth static slot to transmit the attitude control data and ATmega128 uses the eleventh static slot to transmit the position data. Fig. 15 depicts the timing behaviors of the tasks which are similar to the timing diagram in Fig. 13 due to the modified communication configuration which does not affect the computation part. The message from *DataHandler* to *OuterLoop* is transmitted during the eleventh static slot, which can be used as the latest state message by *OuterLoop*. The improvement of the bandwidth increases the cost of the system but stabilizes the control performance which is shown in Fig. 14.

V. Conclusion

In this paper, a simulation framework for design of mixed TT/ET distributed control system is introduced. The framework consists of a virtual platform model in SystemC/TLM and a model transformation approach to generate the virtual platform for a designed system. The virtual platform model has three different models, which are abstract RTOS model, abstract communication system model, and abstract hardware model. The RTOS model is used to capture the dynamic behaviors of TT/ET tasks, and the communication system

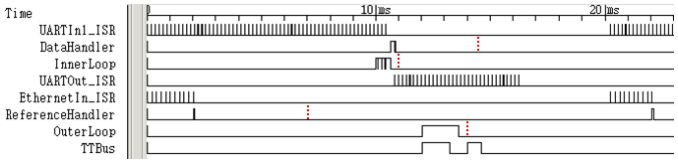


Fig. 13. Timing diagram of the control system with bus bandwidth of 400Kbit/s.

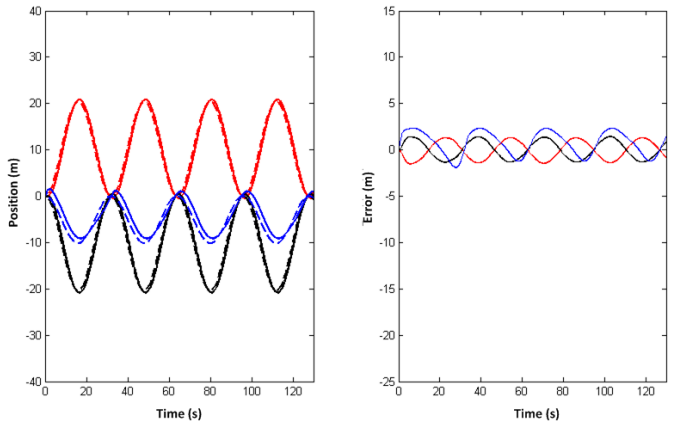


Fig. 14. Control performance with bus bandwidth of 2Mbit/s.

model is used to capture the behaviors of TT/ET communication. The hardware model integrates the RTOS model and the communication system model together. Two model transformations translate the ESMoL model to the corresponding virtual platform model for simulation. We present a case study on a Quadrotor flight control using this framework, and give the simulation results for different bandwidths of the TT/ET buses.

The future work includes introducing a more realistic communication system model with startup, restart, and clock synchronization services and more realistic hardware models that give more realistic timing behaviors.

REFERENCES

[1] Byteflight Homepage. http://www.byteflight.com.

[2] FlexRay Homepage. http://www.flexray.com.

[3] IEEE Standard SystemC Language Reference Manual, 2005.

[4] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. *Embedded System Design: Modeling, Synthesis and Verification*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[5] A. Gerstlauer, H. Yu, and D. D. Gajski. RTOS Modeling for System Level Design. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, 2003.

[6] Z. He, A. Mok, and C. Peng. Timed RTOS Modeling for Embedded System Design. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, RTAS '05, pages 448–457, 2005.

[7] S. Hong, S. Yoo, S. Lee, S. Lee, H. J. Nam, B.-S. Yoo, J. Hwang, D. Song, J. Kim, J. Kim, H. Jin, K.-M. Choi, J.-T. Kong, and S. Eo. Creation and Utilization of A Virtual Platform for Embedded Software Optimization: An Industrial Case Study. In *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '06, pages 235–240, 2006.

[8] Y. Hwang, G. Schirner, S. Abdi, and D. G. Gajski. Accurate Timed RTOS Model for Transaction Level Modeling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1333–1336, 2010.

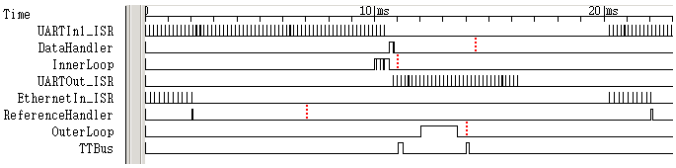


Fig. 15. Timing diagram of the system with bus bandwidth of 2Mbit/s.

[9] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[10] N. Kottenstette and J. Porter. Digital Passive Attitude and Altitude Control Schemes for Quadrotor Aircraft. In *Proceedings of the 7th IEEE Intl. Conf. on Control and Automation*, ICCA '09, ChristChurch, New Zealand, 2009.

[11] R. Le Moigne, O. Pasquier, and J.-P. Calvez. A Generic RTOS Model for Real-time Systems Simulation with SystemC. In *Proceedings of the conference on Design, automation and test in Europe - Volume 3*, DATE '04, pages 30082–, 2004.

[12] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The generic modeling environment. In *Workshop on Intelligent Signal Processing*, 2001.

[13] A. Metzner. Analyzing Mixed Event Triggered/Time Triggered Systems.

[14] K. D. Nguyen, P. S. Thiagarajan, and W.-F. Wong. A UML-Based Design Framework for Time-Triggered Applications. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 39–48, 2007.

[15] S. Park, W. Olds, K. G. Shin, and S. Wang. Integrating Virtual Execution Platform for Accurate Analysis in Distributed Real-Time Control System Development. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 61–72, 2007.

[16] J. Perez, A. Perez, and R. Obermaisser. Executable Time-Triggered Model (E-TTM) for Real-Time Control Systems. In *Proceedings of the 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, ISORC '10, pages 42–49, 2010.

[17] T. Pop, P. Eles, and Z. Peng. Design Optimization of Mixed Time/Event-Triggered Distributed Embedded Systems. In *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '03, pages 83–89, 2003.

[18] T. Pop, P. Eles, and Z. Peng. Schedulability Analysis for Distributed Heterogeneous Time/Event Triggered Real-Time Systems. In *Proceedings of the 15th EuroMicro Conf. on Real-Time Systems*, ECRTS '03, pages 257–266, 2003.

[19] J. Porter and G. Hemingway et al. The ESMoL Language and Tools for High-Confidence Distributed Control Systems Design. Part 1: Language, Framework, and Analysis. Technical Report ISIS-10-109, ISIS, Vanderbilt Univ., 2010.

[20] J. M. Rushby. Bus Architectures for Safety-Critical Embedded Systems. In *Proceedings of the First International Workshop on Embedded Software*, EMSOFT '01, pages 306–323, 2001.

[21] N. Scaife and P. Caspi. Integrating Model-Based Design and Preemptive Scheduling in Mixed Time- and Event-Triggered Systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, ECRTS '04, pages 119–126, 2004.

[22] J. Sztipanovits, X. D. Koutsoukos, G. Karsai, N. Kottenstette, P. J. Antsaklis, V. Gupta, B. Goodwine, J. S. Baras, and S. Wang. Toward a science of cyber-physical system integration. *Proceedings of the IEEE*, 100(1):29–44, 2012.

[23] T. Yokoyama. An Aspect-Oriented Development Method for Embedded Control Systems with Time-Triggered and Event-Triggered Processing. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, RTAS '05, pages 302–311, 2005.

[24] H. Zabel, W. Müller, and A. Gerstlauer. Accurate RTOS Modeling and Analysis with SystemC. In W. Ecker, W. Müller, and R. Dömer, editors, *Hardware-dependent Software*, chapter 9, pages 233–260. Springer Netherlands, 2009.

Notes

Notes

An Analytical Model of the CAN Bus for Online Schedulability Test

Zhenwu Shi and Fumin Zhang, Georgia Institute of Technology

***Abstract*—Controller area network (CAN) is a priority-based bus that supports real-time communication. Existing schedulability analysis for the CAN bus is peformed at the design stage, by assuming that all message information is known in advance. However, in pratice, the CAN bus may run in a dynamic environment, where complete specifications may not be available at the design stage and operational requirements may change at system run-time. In this paper, we develop an analytical model that describes the dynamics of message transmission on the CAN bus. Based on this analytical timing model, we then propose an online test that effectively checks the schedulability of the CAN bus, in the presence of online adjustments of message streams. Simulations show that the online test can accurately report the loss of scheduability on the CAN bus.**

I. Introduction

Controller area network (CAN bus) is a serial bus designed for industrial environments. It was first deployed by the automobile industry in the 1980s for decreasing wiring hardness and complexity among electronic control units in vehicles. During the past thirty years, other industries have gradually adopted the CAN bus and applied it to a variety of areas such as robotics, aircraft, medical equipment, and industrial automation. The CAN bus has an important feature of supporting real-time communication. Each message on the CAN bus is assigned a priority and the transmission of messages follows a deterministic order decided by their prorities [1]. This feature makes the CAN bus particularly suitable for systems with stringent time constraints on the communication.

Related Works The application of the CAN bus requires scheduling analysis to check if all messages can meet their deadlines. The scheduling analysis of the CAN bus has been extensively studied. In 1994, Tindell et al. proposed a basic CAN schedulability analysis [2]–[4]. This result was later recognized by Volvo Car Corporation and successfully used as the theoretical foundation for commercial CAN schedulability analysis tools [5]. However, the basic CAN schedulability analysis in [2]–[4] is based on ideal assumptions of the CAN bus that may not be supported in real applications. Since then, a lot of research has been conducted to improve the basic CAN schedulability analysis. [6]–[9] studied the effect of hardware limitations on the scheduling analysis of the CAN bus. [10]–[12] extended the basic CAN schedulability analysis to account for the transmission errors. [13], [14] studied the schedulability issues of the CAN bus when messages have offsets. [15] proposed a probalistic analysis of the response time of messages on the CAN bus. In 2007, Davis et al. revisted the basic CAN schedulability analysis and corrected some significant flaws [16].

Motivation. Existing schedulability analysis for the CAN bus is performed at the design stage, by assuming that all of message information is known in advance. This assumption used to be valid in the earlier design of embedded systems, which deliberately abstracted away properties of the physical world. However, this is not always be the case. Recent years have witnessed a growing trend for developing embedded systems that are closely integrated with the physical world. For example, Cyber Physical Systems research community has emerged, with the aim of underpinning the integration of cyber and physical elements across all application sectors [17], [18]. However, one of the major design challenges brought by this trend is that the embedded system needs to operate in a dynamic environment, where complete specifications are not possible at the design stage and/or operational requirements may change at system runtime [19]. Particularly concerning the CAN bus in embedded systems, the design challenge means that the CAN bus may frequently experience online adjustment of communication requirements, such as addition, removal, and change of message streams. To guarantee normal operation of the CAN bus in a dynamic environment, we need an online test that can check the schedulability of the CAN bus during system runtime.

Contributation. In this paper, we analytically model the dynamics of message transmission on the CAN bus through a non-block, deterministic hybrid automaton. Then, based on this analytical timing model, we propose an online test that re-evaluates the schedulability of the CAN bus. Our contribution is two manifold: (1) an analytical model of the CAN bus; and (2) an online schedulability test, which is necessary and sufficient, hence gives the least conservative schedulability test.

Organization. The remainder of this paper is organized as follows. Section II presents an overview of the CAN protocol. Section III formulates the problem. Section IV uses a hybrid automaton to analytically describe the dynamics of scheduling messages on the CAN bus. Section V proposed

The research is supported by ONR grants N00014-08-1-1007, N00014-09-1-1074, and N00014-10-10712 (YIP), and NSF grants ECCS-0841195 (CAREER), CNS-0931576, and ECCS-1056253.

Zhenwu Shi and Fumin Zhang are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, USA zwshi, fumin@gatech.edu

an online schedulability test. Section VI uses simulations to show the effectiveness of our online schedulability test.

II. Controller Area Network

This section overviews fundamentals of the CAN protocol, with emphasis on message routing and bus arbitration, which are considered as key features of the CAN protocol. Other part of the CAN protocol can be found in technique documents avaialable online.

A. Message Routing

The CAN protocol defines a standard data message that encapsulates information transmitted between a source node and one or more receivers. As shown in Fig. 1, the data message is composed of seven fields: an SOF field, which represents the start of the message; an identifier field, which is a unique number assigned to the message; a control field, which indicates the length of the data field; a data field, which contains the information encapsulated in the message; a CRC field, which checks the integerity of the message; an ACK field, which acknowledges the reception of the message; and an EOF field, which represents the end of the message.

Figure 1. A standard message frame in CAN

The CAN protocol is a content-based protocol rather than an address-based protocol such as TCP [20]. Unlike the latter, which assigns each message an explicit destination address, the CAN protocol assigns each message a unique identifier. Based on the identifier, the CAN protocol routes messages as follows: When a node attempts to transmit a message, it broadcasts the message on the CAN bus; and each individual node receives the message from the CAN bus and, based on the identifer, decides whether or not to process the message. Such content-based protocol has two major advantages. First, a message can be destined for any number of nodes simultaneously, which increases the utilization of the CAN bus. Second, additional nodes can be added to the existing CAN bus without the necessity to reprogram all other nodes to recognize this addition, which increases the flexibility of the CAN bus.

B. Bus Arbitration

The CAN bus is a serial bus which only allows one node to transmit a message at a time. If two or more nodes attempt to transmit messages at the same time, collisions will happen on the CAN bus. The CAN protocol resolve the collisions through an arbitration scheme known as CSMA/BA (Carrier Sense Multiple Access with Bitwise Arbitration), which uses the identifier of each message as its priority and then based on priorties decides which message may be granted access to the CAN bus.

Using identifiers as priorities is enabled by the wired-AND property of the CAN bus: if multiple nodes are transmitting messages at the same time and one of the nodes transmits a logic bit “0”, the value of the bus will be “0”; and only if all of the nodes transmit a logic bit ”1” will the value of the bus be “1”. Based on this property, CSMA/BA performs arbitration as follows: (1) the arbitration starts from the first bit in the identifier field and ends at the last bit in the identifier field; (2) each node transmits the identifier of a message while monitoring the resulting bus value; and (3) if a node’s transmitted bit differs from the value of the bus, the node detects a collision and aborts its message transmission; if a node’s transmitted bit is same as the value of the bus, the node continues its message transmission. Since each message has a unique identifier, a node transmitting the last bit of the identifier without detecting a collision must be transmitting the highest priority message, and can continue transmitting the remaing part of the message. According to the above arbitration process, we can see that: (1) The logic bit “0” can always win arbitration over the logic bit ”1”, therefore the message with a lower value in the identifier field has a higher priority; (2) the highest-priority message wins arbitration without being disturbed since the transmission of lower priority messages will automatically back off and wait; and (3) the allocation of priorities to messages in identifiers makes the CAN bus particularly suitable for real-time communication.

III. Problem Formulation

We consider a system based on the CAN bus, as illustrated in Figure 2. The CAN bus is shared by multiple nodes. Each node on the CAN bus consists of three parts, where applications generate and utilize messages, host processors carry out user-defined functions, and CAN controllers implement the basic CAN protocol. Each node on the CAN bus may transmit multiple messages to other nodes.

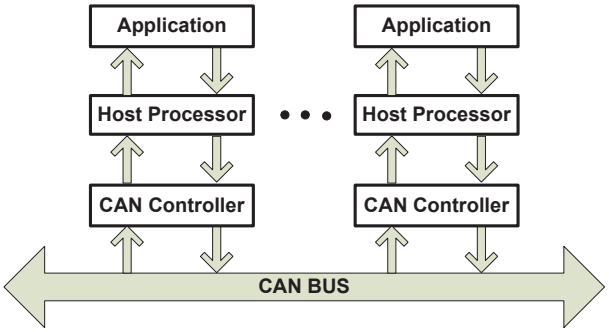


Figure 2. An example of the CAN bus

In this paper, we investigate the problem of operating the CAN bus in a dynamic environment, where communication requirement may frequently change at system run-time, such as on-line addition, removal, and change of message streams.

The online adjustment of message streams may lead to the unschedulability of the CAN bus. Thus, it is necessary to introduce a new way to re-evaluate the schedulability of the CAN bus at system run-time as follows:

Definition 3.1: An online schedulability test over a time interval $[t_a, t_b]$ checks if all message instances are able to meet their deadlines within $[t_a, t_b]$.

As the starting time increases, the time interval $[t_a, t_b]$ will slide forward. The length of the interval $t_b - t_a$ depends on how much into the future we want to perform the schedulability test. All mathematical tools developed in this paper are centered around the online schedulability test within the time interval $[t_a, t_b]$.

A. System Configuration

There are many different ways of desigining the architecture of the CAN bus. Each design may lead to a different model. In this paper, we assume that the CAN bus in Figure 2 satisfies the following conditions

1) The CAN bus is reliable such that no error exists in the transmission;
2) At each node, among all messages that are ready for transmission, the message with the highest priority will be always transmitted first.

The above conditions are conventional assumptions that have been widely used in the research community of the CAN bus. As a first step towards the analytical modeling of the CAN bus, we rely on them for our derivation. While the above conditions may not be satisfied in all application scenarios, we believe that refined modeling can be carried out under the same framework .

Moreover, we configure the CAN bus such that each node will automatically drop off transmission requests of messages that cannot meet their deadlines.

B. Model and Notations

We describe a system model for the CAN bus and several notations that will be used in the rest of the paper.

Consider a set of independent messages $\{\tau_1, \cdots, \tau_N\}$ transmitting on the CAN bus within the time interval $[t_a, t_b]$. Each message τ_n is characterized by a tuple $\{a_n^k, C_n^k, T_n^k, E_n^k, P_n^k\}$ defined as follows:

- a_n^k: the release time of the the k-th instance of τ_n;
- C_n^k: the transmission time of the k-th instance of τ_n;
- T_n^k: the inter-release time, i.e. $T_n^k = a_n^{k+1} - a_n^k$;
- E_n^k: the relative deadline of the k-th instance of τ_n;
- P_n^k: the identifier of the k-th instance of τ_n;

Note that the characteristics of τ_n is defined at the instance level. Such definition allows each instance of τ_n to have different characteristics, which makes our model applicable to not only periodic messages but also non-periodic messages.

Remark 3.2: Since the CAN bus operates in a environment where the communication requirement may frequently change, the set of messages $\{\tau_1, \cdots, \tau_N\}$ may dynamically adjust at runtime.

IV. An analytical timing model for scheduling on the CAN bus

In last section, we have established necessary notations to describe a set of messages on the CAN bus. In this section, we will develop an analytical timing model that describes the dynamics of scheduling messages on the CAN bus.

A. State Variables

To describe how message $\{\tau_1, \cdots, \tau_N\}$ are scheduled on the CAN bus from a dynamic system point of view, we introduce a state vector $Z(t) = [D(t), R(t), O(t)]$.

Definition 4.1: The first component $D(t)$ is defined as a vector $D(t) = [d_1(t), \cdots, d_N(t)]$, where $d_n(t)$, for $n = 1, 2, \cdots, N$, denotes how long after t the next instance of τ_n will be released.

Definition 4.2: The second component $R(t)$ is defined as a vector $R(t) = [r_1(t), ..., r_N(t)]$, where $r_n(t)$, for $n = 1, 2, ..., N$, denotes the remaining transmission time of the current instance of τ_n after time t.

Definition 4.3: The third component $O(t)$ is defined as a vector $O(t) = [o_1(t), ..., o_N(t)]$, where $o_n(t)$, for $n = 1, 2, ..., N$, denotes how much time has elapsed before the current instance of τ_n finishes transmission.

With the state vector well defined, we can study the dynamics of scheduling $\{\tau_1, \cdots, \tau_N\}$ on the CAN bus through the evolution of $Z(t)$.

B. Evolution of State Vector

The state vector $Z(t)$ evolve continuously most of the time, except when two "special" events happen. One special event is the arrival of a new instance. When this event happens, the value of $Z(t)$ is reset to the characteristics of the new instance. The other special event is that a message finishes transmission on the CAN bus and another message starts transmission. When this event happens, the evolution dynamics of $Z(t)$ switches discontinuously. Since the state vector $Z(t)$ exhibits both continuous and discrete dynamic behaviors, the evolution of $Z(t)$ can be described by a hybrid automaton defined as follows.

Definition 4.4: A hybrid automaton that describes the dynamics of scheduling $\{\tau_1, \cdots, \tau_N\}$ is a collection $H = \{Q, Z, F, \mathrm{Dom}, \mathrm{Edge}, \mathrm{Guard}, \mathrm{Reset}\}$ where

- $Q = \{q_0, q_1, \cdots, q_N\}$ is a set of modes, where the mode q_0 indicates that no message is being transmitted on the CAN bus and the mode q_i $(1 \leq i \leq N)$ indicates that τ_i is being transmitted on the CAN bus;

- $Z(t) = [D(t), R(t), O(t)] \in \mathbb{R}^{3N}$ is a continuous state vector as defined above;
- $F : Q \times Z \to \mathbb{R}^{3N}$ is the flow map. For each $q_i \in Q$, $F(q_i, Z)$ describes the continuous evolution of Z in the mode q_i;
- $\mathrm{Dom} : Q \to 2^Z$ is the domain of modes. For each $q_i \in Q$, $\mathrm{Dom}(q_i)$ identifies a set of Z that evolves continuously in the mode q_i;
- $\mathrm{Edge} : \mathrm{Edge} \subseteq Q \times Q$ is a set of edges. Each $(q_i, q_j) \in \mathrm{Edge}$ indicates that a discrete transition from the mode q_i to the mode q_j is possible;
- $\mathrm{Guard} : \mathrm{Edge} \to 2^Z$ is the jump condition. For each $(q_i, q_j) \in \mathrm{Edge}$, $\mathrm{Guard}(q_i, q_j)$ identifies a set of Z that can trigger a discrete transition from the mode q_i to the mode q_j;
- $\mathrm{Reset} : \mathrm{Edge} \times Z \to 2^Z$ is the reset map. For each $(q_i, q_j) \in \mathrm{Edge}$, $\mathrm{Reset}(q_i, q_j, Z)$ describes the value to which Z is reset during a discrete transition from the mode q_i to the mode q_j;

Figure 3 demonstrates a directed graph representation of the hybrid automaton H when two independent messages $\{\tau_1, \tau_2\}$ are being transmitted on the CAN bus. The graphical representation of the H with other values of N can be easily constructed using the same methodology. As shown in Figure 3, the vertices represent modes and the arrows represent edges. Within each vertex the flow map and the domain set are indicated. Along each edge the jump condition and the reset map are shown.

In the following part of this section, we will derive the expressions of F, Dom, Edge, Guard and Reset, respectively.

1) **Flow Map:** We discuss the continuous evolution of $Z(t)$ in any mode $q_i \in Q$. Since $Z(t)$ consists of three components as $Z(t) = [D(t), R(t), O(t)]$, we will discuss the continuous evolution of each component respectively. We use $\Delta t > 0$ to denote an arbitrarily small change in time.

First, we study the continuous evolution of $D(t)$ in the mode q_i. Consider an element $d_n(t) \in D(t)$. According to Definition 4.1, we know that $d_n(t)$ will continuously decrease within $[t, t + \Delta t]$, i.e.

$$d_n(t + \Delta t) = d_n(t) - \Delta t \tag{1}$$

which implies the continuous evolution of $d_n(t)$ as

$$\dot{d}_n(t) = \lim_{\Delta t \to 0} \frac{d_n(t + \Delta t) - d_n(t)}{\Delta t} = -1 \tag{2}$$

Therefore, we have the continuous evolution of $Q(t)$ in the mode q_i as

$$\dot{D}(t) = [-1, \cdots, -1]^T \tag{3}$$

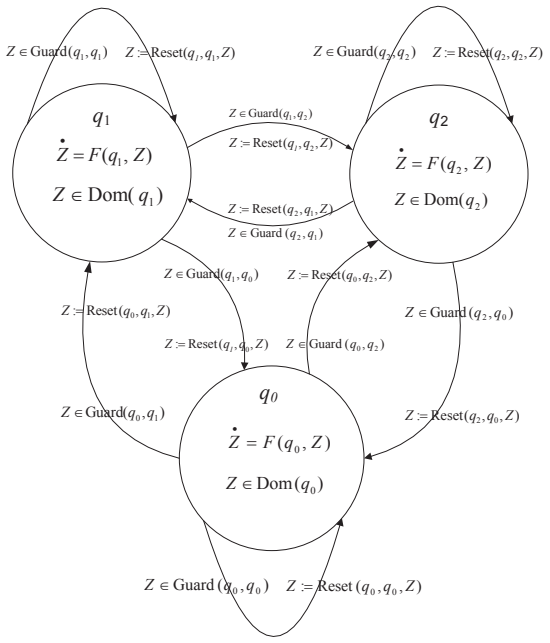


Figure 3. Graphical representation of the hybrid automaton H when $N = 2$

Next, we study the continuous evolution of $R(t)$ in the mode q_i. Consider an element $d_n(t) \in R(t)$. According to Definition 4.2, we know that $r_n(t)$ will decrease within $[t, t + \Delta t]$ when τ_n is being transmitted at time t, and keeps constant otherwise. Moreover, only τ_i can be transmitted in the mode q_i, as stated in Definition 4.4. Therefore, we have that

$$r_n(t + \Delta t) = \begin{cases} r_n(t) - \Delta t & n = i \\ r_n(t) & \text{otherwise} \end{cases} \tag{4}$$

which implies the continuous evolution of $r_n(t)$ as

$$\dot{r}_n(t) = \begin{cases} -1 & n = i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Therefore, we can express the continuous evolution of $R(t)$ in the mode q_i as

$$\dot{R}(t) = [0, \cdots, -1, \cdots, 0]^T \tag{6}$$

where -1 is in the i-th entry of the vector.

Finally, we study the continuous evolution of $O(t)$ in the mode q_i. Consider an element $o_n(t) \in O(t)$. According to Definition 4.3, we know that the evolution of $o_n(t)$ depends on whether the current instance of τ_n has finished transmission. If the current instance of τ_n has finished transmission before time t, i.e. $r_n(t) = 0$, $o_n(t)$ will not increase within $[t, t + \Delta t]$. On the other hand, if the current instance of τ_n has not finished transmission before t, i.e. $r_n(t) > 0$, $o_n(t)$ will increase within $[t, t + \Delta t]$. By defining a function sgn such that $\mathrm{sgn}(x) = 1$ when $x > 0$, $\mathrm{sgn}(x) = 0$ when $x = 0$,

and $\mathrm{sgn}(x) = 1$ when $x < 0$, we have that

$$o_n(t + \Delta t) = o_n(t) + \mathrm{sgn}(r_n(t))\,\Delta t \tag{7}$$

which implies the continuous evolution of $o_n(t)$ as

$$\dot{o}_n(t) = \lim_{\Delta t \to 0} \frac{o_n(t + \Delta t) - o_n(t)}{\Delta t} = \mathrm{sgn}(r_n(t)) \tag{8}$$

Therefore, we have the continuous evolution of $O(t)$ in the mode q_i as

$$\dot{O}(t) = [\mathrm{sgn}(r_1(t)), \cdots, \mathrm{sgn}(r_N(t))]^T \tag{9}$$

In summary, equation (3), (6) and (9) constitute the continuous evolution of $Z(t)$ in any mode $q_i \in Q$.

2) **Domain of Modes:** For the ease of expression, we define an auxiliary variable $G(t)$ as follows

Definition 4.5: $G(t)$ is a set of indices of messages, which are active for transmission at time t.

$$G(t) = \left\{n|\ r_n(t) = C_n^k \text{ and } o_n(t) + C_n^k \le E_n^k\right\} \tag{10}$$

where k represents the index of the current instance of τ_n at time t, $r_n(t) = C_n^k$ specifies messages that have not transmitted yet, and $o_n(t) + C_n^k < E_n^k$ specifies messages that can meet their deadlines.

In the mode q_0, the state Z will continuously evolve as long as the following two conditions are both satisfied: no new instance of $\{\tau_1, \cdots, \tau_N\}$ is released and no message is active for transmission on the CAN bus. To meet the first condition, we have that

$$\min_{1 \le n \le N} \{D(t)\} > 0 \tag{11}$$

where Definition 4.1 is applied. To meet the second condition, we have that

$$\mathrm{Card}\left(G(t)\right) = 0 \tag{12}$$

where $\mathrm{Card}(\cdot)$ is a cardinality function that measures the number of elements in a set. Therefore, we have the domain of the mode q_0 as

$$\mathrm{Dom}(q_0) = \left\{ Z(t)\,|\, \min_{1 \le n \le N} \{D(t)\} > 0 \text{ and } \mathrm{Card}(G(t)) = 0 \right\} \tag{13}$$

In any other mode q_i where $1 \le i \le N$, the state Z will continuously evolve as long as the following two conditions are both satisfied: no new instance of $\{\tau_1, \cdots, \tau_N\}$ is released and τ_i is being transmitted on the CAN bus. To meet the first condition, we have that

$$\min_{1 \le n \le N} \{D(t)\} > 0 \tag{14}$$

where Definition 4.1 is applied. To meet the second condition, we have that

$$0 < r_i(t) \le C_i^k \tag{15}$$

Therefore, we have the domain of the mode q_i where $1 \le i \le N$ as

$$\mathrm{Dom}(q_i) = \left\{ Z(t)|_{\min_{1 \le n \le N} \{D(t)\} > 0 \text{ and } 0 < r_i(t) \le C_i^k} \right\} \tag{16}$$

3) **Edges and Jump Conditions:** According to the definition of Q, we know that the transition between any two modes is possible. Therefore, we have that

$$\mathrm{Edge} = Q \times Q \tag{17}$$

where an edge $(q_i, q_j) \in \mathrm{Edge}$ represents a transition from the mode q_i to the mode q_j.

For any edge $(q_i, q_j) \in \mathrm{Edge}$, we discuss the jump condition $\mathrm{Guard}(q_i, q_j)$. Our discussion on jump conditions can be classified into four cases according to different edges.

Case 1: an edge where $i = j$. This transition is triggered when any message in $\{\tau_1, \cdots, \tau_N\}$ release a new instance, i.e.

$$\mathrm{Guard}(q_i, q_i) = \{ \min_{1 \le n \le N} \{D(t)\} = 0 \} \tag{18}$$

Case 2: an edge where $i \ne j$ and $1 \le i, j \le N$. This transition is triggered when τ_i finishes transmission and τ_j starts transmission. Thus, we have

$$\mathrm{Guard}(q_i, q_j) = \{r_i(t) = 0 \text{ and } j = \operatorname*{argmin}_{n \in G(t)} P_n^k\} \tag{19}$$

where $r_i(t) = 0$ indicates that τ_i has finished transmission on the CAN bus at time t and $j = \mathrm{argmin}_{n \in G(t)} P_n^k$ indicates that τ_j has the highest priority among all messages that are active for transmission at time t.

Case 3: an edge where $1 \le i \le N$ and $j = 0$. This transition is triggered when τ_i finishes transmission and no message is active for transmission at time t. Thus, we have

$$\mathrm{Guard}(q_i, q_0) = \{r_i(t) = 0 \text{ and } \mathrm{Card}\left(G(t)\right) = 0\} \tag{20}$$

where $r_i(t) = 0$ indicates that τ_i has finished transmission at time t and $\mathrm{Card}\left(G(t)\right) = 0$ indicates that no message is active for transmission at time t.

Case 4: an edge where $i = 0$ and $1 \le j \le N$. This transition is triggered when τ_i starts its transmission after the CAN bus has been idle for a while.

$$\mathrm{Guard}(q_0, q_j) = \{\mathrm{Card}\left(G(t)\right) > 0 \text{ and } j = \operatorname*{argmin}_{n \in G(t)} P_n^k\} \tag{21}$$

where $\mathrm{Card}\left(G(t)\right) > 0$ indicates that there are messages active for transmission on the CAN bus at time t and $j = \mathrm{argmin}_{n \in G(t)} P_n^k$ indicates that τ_j has the highest priority among all messages that are active for transmission at time t.

4) **Reset Map:** We use t^+ to denote the time right after the reset.

We first discuss the reset map for an edge (q_i, q_j) where $i = j$. As discussed in equation (18), this transition happens when a new instance of $\{\tau_1, \cdots, \tau_N\}$ is released. Consider a message τ_n.

Case 1: if the new instance released at time t is not from τ_n, i.e. $d_n(t) > 0$. In this case, the state variables of τ_n hold their values during the transition, i.e.

$$\begin{gathered} \text{if } \ d_n(t) > 0, \text{we have :} \\ d_n(t^+) = d_n(t) \ \ r_n(t^+) = r_n(t) \ \ o_n(t^+) = o_n(t) \end{gathered} \tag{22}$$

Case 2: if the new instance released at time t is from τ_n, i.e. $d_n(t) = 0$. In this case, the state variables of τ_n is reset to the characteristics of the new instance.

$$\begin{gathered} \text{if } \ d_n(t) = 0, \text{we have :} \\ d_n(t^+) = T_n^{k+1} \ \ r_n(t^+) = C_n^{k+1} \ \ o_n(t^+) = 0 \end{gathered} \tag{23}$$

Applying equation (22) and (23) for $n = 1, \cdots, N$, we have the reset map $\mathrm{Reset}(q_i, q_j, Z)$ where $i = j$.

Next, we discuss the reset map for an edge (q_i, q_j), where $i \neq j$. During this transition, the state vector $Z(t)$ remains constant. Thus, $\mathrm{Reset}(q_i, q_j, Z)$ equals to an identity map:

$$\mathrm{Reset}(q_i, q_j, Z) = Z \tag{24}$$

V. Online Schedulability Test

In this section, we show how to perform an online schedulability test of the CAN bus by utilizing the analytical timing model developed in Section IV.

A. A Necessary and Sufficient Condition for Schedulability

Consider a set of messages $\{\tau_1, \cdots, \tau_N\}$ being transmitted on the CAN bus within the time interval $[t_a, t_b]$. The online schedulability test over $[t_a, t_b]$ can be decomposed to check whether each message is able to meet its deadlines within $[t_a, t_b]$. The CAN bus is schedulable within $[t_a, t_b]$ if and only if all messages are schedulable within $[t_a, t_b]$. The following theorem states the necessary and sufficient condition for the schedulability of τ_n within $[t_a, t_b]$.

Theorem 5.1: A message τ_n is schedulable within $[t_a, t_b]$ if and only if for all time points $t \in [t_a, t_b]$ such that $d_n(t) = 0$, we have that $r_n(t) = 0$.

Proof: According to Definition 4.1, we know that $d_n(t) = 0$ implies that an new instance of τ_n is released at time t. At this time point, the state variables of τ_n represent the final status of the old instance. Hence, the old instance of τ_n can meet its deadline if and only if the remaning transmission time is zero, i.e.

$$r_n(t) = 0$$

■

B. Initial State and Message Characteristics

At any time t_a, running the analytical timing model of the CAN bus within $[t_a, t_b]$ requires informaiton of the initial state and message characteristics.

First, we discuss the reconstruction of the initial state $[Q(t_a), Z(t_a)]$. For $Q(t_a)$, since messages are broadcast on the CAN bus, each node can easily detect which message is being transmitted on the CAN bus. For $Z(t_a)$, according to Definition 4.1, 4.2, and 4.3, its value depends on two types of information: characteristics of current instances of $\{\tau_1, \cdots, \tau_N\}$, and how these instances have transmitted from time of release till time t_a. The first type of information has already known at the time when an instance is released. The second type of information can be obtained from software tools that monitor and record data traffic on the CAN bus. For example, Esd Electronics, Inc. provides a monitoring tool as integral part of the driver for their CAN controllers.

Next, we consider message characteristics. In this paper, we assume that message characteristics within $[t_a, t_b]$ is known at time t_a. This assumption is reasonable as it is possible to predict a little bit ahead in the future. Also, this is the weakest assumption for performing any online schdulability test.

C. Implementation of the online schedulability test

At any time t_a, given the initial state variable and message characteristics within $[t_a, t_b]$, we can perform the online schedulability test over the time interval $[t_a, t_b]$ using Algorithm 1. This algorithm iteratively checks the schedulability of the CAN bus in the following ways: (1) within each discrete mode, it allows the state vector $Z(t)$ to continuously evolve according to the flow map F until $Z(t)$ reaches the boundary of the discrete domain and triggers a transition, as indicated by Line 8 and 9; (2) if the transition is from a mode to itself, it evaluates the schedulability of τ_n according to Theorem 5.1, as shown in Lines $11 - 15$; and (3) if the transition is between two different modes, it re-calcuates the destination mode, as indicated by Line 18.

The variable DS_n indicates the online schedulability test result of τ_n within $[t_a, t_b]$: when an instance of τ_n is schedulable, its corresponding element in DS_n equals to 0; otherwise, its corresponding element in DS_n equals to 1. A message τ_n is schedulable within $[t_a, t_b]$ if and only if all instances of τ_n that are released within $[t_a, t_b]$ are schedulable, i.e. $\max\{\mathrm{DS_n}\} = 0$. The CAN bus is schedulable within $[t_a, t_b]$ if and only if all messages are schedulable within $[t_a, t_b]$, i.e. $\max_{1 \leq n \leq N}\{\max\{\mathrm{DS_n}\}\} = 0$.

VI. Simulation

In this section, we use a simple example to demonstrate the effectiveness of our online schedulability test when

Algorithm 1: Online Schedulability Test

/*Schedulability of the CAN bus within $[t_a, t_b]$ */
Data: t_a, t_b, $Q(t_a^-)$, $Z(t_a^-)$, $\{C_n^k, T_n^k, E_n^k, P_n^k\}_{n=1}^N$
Result: $\{\mathrm{DS_n}\}_{\mathrm{n=1}}^{\mathrm{N}}$

$t = t_a$; mode $= Q(t_a^-)$;
for $n = 1$ **to** N **do**
  $\mathrm{DS_n} = [\,]$;
while $t < t_b$ **do**
  switch mode **do**
    ……;
    case n :
      while $Z(t) \in \mathrm{Dom}(q_n)$ **do**
        $Z(t) = F(q_n, Z(t))$;
      if $\min\limits_{1 \le n \le N}\{D(t)\} == 0$ **then**
        for $n = 1$ **to** N **do**
          if $d_n(t) == 0$ **and** $r_n(t) == 0$ **then**
            $\mathrm{DS_n} = [\mathrm{DS_n}\ \ 0]$;
          else if $d_n(t) == 0$ **and** $r_n(t) > 0$ **then**
            $\mathrm{DS_n} = [\mathrm{DS_n}\ \ 1]$;
        $Z(t) = \mathrm{Reset}(q_n, q_n, Z(t))$;
      else
        Re-calculate the value of mode ;
    ……;
return $\{\mathrm{DS_n}\}_{\mathrm{n=1}}^{\mathrm{N}}$;

message streams on the CAN bus frequently changes. Our methods can be easily applied to more complex examples.

At the design stage, we assume that three periodic messages $\{\tau_1, \tau_2, \tau_3\}$ start their transmission on the CAN bus simultaneously from time 0. The three messages have the following characteristics

$$\begin{aligned}
\left[\ T_1^k,\ T_2^k,\ T_3^k\ \right] &= [\ 200,\ 250,\ 300\ ]\ \mathrm{ms},\\
\left[\ C_1^k,\ C_2^k,\ C_3^k\ \right] &= [\ \ 40\ ,\ \ 50\ ,\ \ 60\ ]\ \mathrm{ms},\\
\left[\ E_1^k,\ E_2^k,\ E_3^k\ \right] &= [\ 100,\ 175,\ 200\ ]\ \mathrm{ms}.
\end{aligned}$$

Moreover, the three messages are assigned unique identifiers such that

$$P_1^k < P_2^k < P_3^k$$

which implies that τ_1 has a higher priority than τ_2, which has a higher priority than τ_3.

At system run-time, we consider two types of online communication adjustments. One is the change of message periods. We assume that τ_1 and τ_2 change their periods within $[22, 26]$s as

$$\left[\ T_1^k,\ T_2^k\ \right] = [160,\ 210]\ \mathrm{ms}.$$

The other type of online adjustment is the addition of new messages. We assume that another periodic messages τ_4 appears on the CAN bus within $[20, 26]$s. The new message has the following characteristics

$$\left[\ T_4^k,\ C_4^k,\ E_4^k\ \right] = [\ 200,\ 50,\ 200]\ \mathrm{ms},$$

Moreover, the new message is assigned a unique identifier such that

$$P_1^k < P_4^k < P_2^k < P_3^k$$

Figure 4(a) shows the transmission of four messages on the CAN bus within the time interval $[20.5, 23.5]$s. The value "0.5" indicates that the transmission of the message is blocked by other higher priority messages on the CAN bus; the value "1" indicates that the message is being transmitted on the CAN bus; and the value "0" indicates that the message finishes transmission. By closely examining Figure 4(a), we can see that two instances of τ_3 that are released at time 21s and time 22.8s have not been transmitted before their deadlines. Figure 4(a) shows the online schedulability test of the CAN bus within the time interval $[20.5, 23.5]$. As discussed in Section V, the value "1" indicates that an instance fails to meet its deadline and the value "0" indicates that an instance meets its deadline. According to Figure 4(b), we can easily see that two instances of τ_3 that are released at time 21s and time 22.8s fail to meet their deadlines. Therefore, the observation in Figure 4(b) exactly match that in Figure 4(a), which implies that our online schedulability test can accurately identify the unschedulable message instances.

VII. Conclusion

This paper proposes an online schedulability test of the CAN bus, which is based on an analytical model. The simulations show that the online schedulability test can accurately report the lost of schedulability on the CAN bus.

References

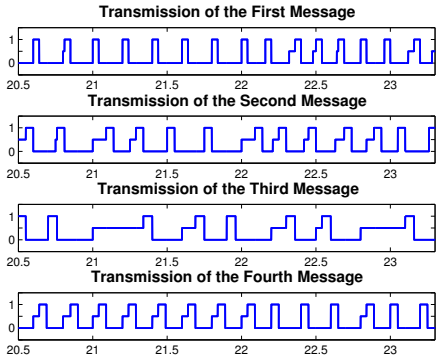
[1] R. B. Gmbh, "CAN Specification Version 2.0," Stuttgart, Germany, Tech. Rep., 1991.

[2] K. W. Tindell, H. Hansson, and A. J. Wellings, "Analysing real-time communications: controller area network (CAN)," in *Proc. of IEEE International Conference on Real-Time Systems Symposium*, Dec. 1994, pp. 259–263.

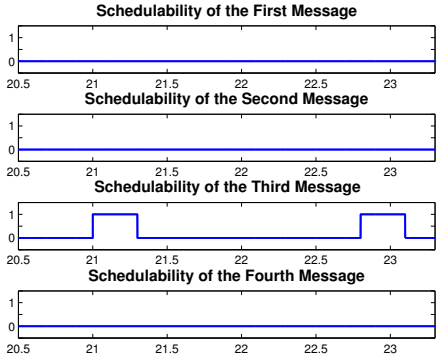
[3] K. Tindell and A. Burns, "Guarantee Message Latency on Control Area Network(CAN)," in *Proc. of International CAN Conference*, 1994, pp. 1–11.

[4] K. Tindell, A. Burns, and A. Wellings, "Calculating Controller Area Network (CAN) message response times," *Control Engineering Practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

[5] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg, "Volcano-a revolution in on-board communications," Volvo Technology Report, Tech. Rep., 1998.



(a) Transmission of Messages



(b) Online Schedulability Test Result

Figure 4. The dynamics of the CAN bus within $[20.523.3]$s

[6] M. D. Natale, "Evaluating message transmission times in Controller Area Networks without buffer preemption," in *Proc. of Brazilian Workshop on Real-Time Systems*, 2006.

[7] D. A. Khan and R. J. Bril, "Integrating hardware limitations in CAN schedulability analysis," in *Proc. of IEEE International Workshop on Factory Communication Systems*, 2010, pp. 207–210.

[8] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Controller Area Network (CAN) Schedulability Analysis with FIFO queues," in *Proc. of Euromicro Conference on Real-Time Systems*, 2011, pp. 45–56.

[9] D. A. Khan, R. I. Davis, and N. Navet, "Schedulability analysis of CAN with Non-abortable Transmission requests," in *Proc. of IEEE International Conference on Emerging Technologies and Factory Automation*, Sept. 2011, pp. 1–8.

[10] I. Broster and A. Burns, "Probabilistic analysis of CAN with faults," in *Proc. of 23rd IEEE Real-Time Systems Symposium*, 2002, pp. 269–278.

[11] S. Punnekkat, H. Hansson, and C. Norstrom, "Response time analysis under errors for CAN," in *Proc. of IEEE Real-Time Technology and Applications Symposium*, 2000, pp. 258–265.

[12] I. Broster, A. Burns, and G. RodrÍguez-Navas, "Timing Analysis of Real-Time Communication under Electromagnetic Interference," *Real-Time Systems*, vol. 30, pp. 55–81, 2005.

[13] M. Grenier, L. Havet, and N. Navet, "Pushing the limits of CAN-scheduling frames with offsets provides a major performance boost," in *Proc. of European Congress Embedded Real Time Software*, 2008.

[14] P. M. Yomsi, D. Bertrand, N. Navet, and R. I. Davis, "Controller Area Network (CAN): Response Time Analysis with Offsets," in *Proc. of IEEE International Workshop on Factory Communication Systems*, 2012, pp. 43–52.

[15] H. Zeng, M. D. Natale, P. Giusto, and A. Sangiovanni-Vincentelli, "Stochastic Analysis of CAN-Based Real-Time Automotive Systems," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 4, pp. 388–401, Nov. 2009.

[16] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[17] W. Wolf, "Cyber-physical systems," *Computers*, vol. 42, no. 3, pp. 88–89, 2009.

[18] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, *Machine Learning in Cyber Trust Security, Privacy, and Reliability*. Springer, 2009, pp. 3–13.

[19] E. A. Lee, "Cyber Physical Systems : Design Challenges," in *Proc. of IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 2008, pp. 363–369.

[20] K. Pazul, "Controller Area Network (CAN) Basics," Microchip Technology Inc., Tech. Rep., 1999.

Notes

Notes

Online Construction of Analytical Prediction Models for Physical Environments: Application to Traffic Scene Modeling

Anurag Umbarkar, *Student Member, IEEE*, Shreyas K Rajagopal and Alex Doboli, *Senior Member, IEEE*
Department of Electrical and Computer Engineering
State University of New York at Stony Brook
Stony Brook, New York, USA
Email: adoboli@ece.sunysb.edu

***Abstract*—This paper presents a methodology to model the dynamics of traffic scenes, including the participating vehicles, vehicle clusters, the attributes and relations of the scene elements, and related events, like cluster merging and splitting. Compared to other methods, this methodology constructs the models online using data coming from sensors. The main steps are to identify the elements of a scene, to find the relations among the elements, and to construct analytical prediction models for the traffic scene dynamics. The paper discusses all the related theoretical aspects, including ontologies for traffic scene description, stochastic prediction of event sequences, and vehicle and cluster identification using sound-based vehicle localization.**

I. Introduction

Modern applications, e.g., intelligent traffic systems, smart power grid, and critical infrastructure monitoring, require large-scale decision making networks that operate in tight interactions with the natural environments. Natural environments are fundamentally different than engineered systems (i.e. plants, autovehicles, and consumer goods), which have been the traditional beneficiaries of optimized control. Engineered systems have, in general, a well-defined and predictable behavior as their operating conditions and requirements are well defined. Natural environments are arguably more complex and diverse with respect to their composing elements and interactions. Many interactions and conditions of the natural world are unknown until they are produced, hence are hard to predict and characterize a-priori. Second, natural environments are continuously changing without necessarily moving towards an end state or progressing towards a final goal. The two arguments stress that the modeling and representation of natural environments must tackle dynamically changing situations that include a large variety of emerging entities and interactions. Precise modeling of traffic scenarios, a particular case of natural environments, is important for the design and optimization of many traffic related applications, including intelligent traffic systems and smart vehicles.

Mathematical models to predict various traffic scenarios have been a researched over the last 60 years [13], [17], [18]. More recently, stochastic modeling methods have been a popular way of representing traffic flows. Belomestny et. al. [3] discuss highway traffic modeling using Gaussian densities. The models give good predictions if the traffic load is light, hence vehicles' interactions are limited. For heavy traffic, however, the model fails to capture well the interactions. A different approach uses conditional autoregressive modeling, in which the local interactions between vehicles follow the characteristics of Markovian process [7]. Ridge regression [19] and Bayesian networks [15] have been also suggested for traffic modeling. Another approach is to use cellular automata [14]. Monte Carlo simulations are used to find possible outcomes of a given scenario, including any outlier conditions. A similar simulation paradigm is pursued also in microscopic traffic models [2], [11]. Microscopic simulators mimic vehicles as particles, such as the particles of a gas [18]. They capture well the local interactions between vehicles [18], [8], [17] but often neglect their physical characteristics, like length and mass. A third approach is to represent traffic flow similar to the dynamics of flowing fluids [2], [13]. A novelty of the technique in [2] is in that it attempts to incorporate driver-specific factors, such as the psychological element. Finally, a fourth approach is to represent traffic similar to queuing systems [9], [10], [12]. They capture well situations, like traffic congestions and the presence of traffic lights.

Modeling and representing traffic scenes remains a challenging effort because of the following three main aspects:

- *Incomplete sensing*: The utilized sensing devices might be insufficient to sample all signals that are relevant in certain conditions. The traditional philosophy of embedded system design is to introduce specific sensors to directly sample all the signals important in every situation. However, the unknown conditions invalidate this strategy as new conditions might involve signals that were not identified a-priori. Besides, constraints, like cost, energy and power consumption, and availability of the hardware resources needed to implement the frontends, limits the number and type of sensors that can be used in a practical design, and hence implicitly the kind of situations in which the embedded node operates efficiently. Also, some of the relevant attributes cannot be directly sensed.
- *Algorithmic limitations*: The algorithmic descriptions in

embedded systems usually express the sequence of steps that transform state values (e.g., variables) for specific inputs and events. An important caveat is that the system is completely defined, meaning that all its state variables, inputs, and expected responses are well defined for all situations. However, this is insufficient as the identity and defining attributes of the elements in a traffic scene are often not known at the time of specification development. Also, the variety of interacting objects is huge and the importance of interactions can change dramatically for specific conditions. New cause-effect schemes can emerge so that new kinds of correlations are induced between the objects. This makes it difficult to use algorithmic descriptions to represent an environment.

- *Human and social dimension*: The importance of these elements depends to a significant degree on the specific interactions among individuals and groups, and how their behavior adjusts to such interactions. For example, individual drivers might decide to adjust their behavior and expectations based on specific external conditions (e.g., weather, time, and so on), their subjective state (i.e. state of emotions), and the behavior of the other individuals (including collaborative and competitive interactions with other individuals). The nature and quality of decision making depends to a large degree on factors that must be analyzed for the specific scenario.

This discussion suggests that modeling natural environments in which CPS operate must first understand the semantics of a given scenario, and then produce prediction models that capture the possible outcomes of the models. Scenario understanding involves identifying the components of a scene, the attributes of the components, the interactions between components, and their dynamics in time.

This paper proposes a modeling methodology to express the dynamics of traffic scenes, including the vehicles and vehicle clusters forming the scene, the attributes and relations among the scene elements, and events, such as cluster merging and splitting. Compared to other methods, this methodology constructs the models online using data coming from sensors. The methodology has three steps: identifying the elements of a scene, finding the relations among elements, and constructing prediction models for the traffic scene dynamics. Scene elements are identified using an ontology describing the possible components, and an algorithm that clusters instances (e.g., vehicles) using the common attributes of the instances and separates existing clusters using the distinguishing attributes. The clustering algorithm is based on Support Vector Machines (SVM). Prediction models are built by relating the analytical expressions of the identified scene elements (the analytical expressions are part of the ontology) through a stochastic scheme which predicts the likelihood of having various event sequences, like cluster merging and splitting. Cluster identification (using SVMs) using sound-based vehicle localization is also discussed in the paper.

The paper has the following structure. Section II discusses the ontology used in representing traffic situations and then suggests a method to construct the representation for a real traffic environment. Section III presents the procedure to estimate the dynamics of certain traffic events, like vehicle cluster merging and splitting. Section IV introduces our solution for detecting clusters and events based on sensor readings. Conclusions end the paper.

II. Problem Description

The goal of this work is to understand at run time the semantics of traffic scenes (environments) based on audio range inputs collected through a network of embedded nodes with sound processing features [1]. Understanding traffic scenes includes the following main challenges:

- *Finding the components of a scene*: This capability identifies the elements of a traffic scene and their defining attributes. The elements include not only physical objects (e.g., objects with attributes directly sensed through the sensors) but also more abstract elements that are used in the reasoning process, like concepts which are not directly sensed but impact the observed signals as well as abstract concepts and categories. Scene components are characterized by a set of well defined, repeatable attributes (which creates the invariant identity of a component) and a set of attributes that distinguish the concept from other concepts.
- *Understanding the relations between the components in a scene*: This capability finds the interdependencies and correlations that exist between the components in a scene, including cause - effect relations, in which a certain element causes or enables a given effect, and various kinds of correlations between elements.
  Getting insight into the cause of the existing relations is a first main requirement. In addition to the correlations that result directly from the nature of the application, other correlations are produced due to specific conditions and properties of the participating elements. For example, traffic flow can be obstructed by an obstacle on the road (direct cause) or a set of drivers with specific driving profiles that slow each other down. The second situation can be inferred from the scene characteristics even if it is not directly specified as a cause for slow traffic.
  Disambiguation is a second main requirement as multiple causes can produce similar effects. For example, group of vehicles slowing down can be either because of some conservative drivers or due to potholes present in the road. The sensed information must be used to deduce the more likely cause that produces a situation among the two possibilities.
- *Predicting the evolution of a scene*: The capability refers to the dynamics (evolution) of a traffic scene, including the possible situations that can emerge within a future time window. Predictions are important to correct erroneous data from the sensors, to identify the necessary and sufficient data needed for scene understanding, and to preemptively adopt decisions for situations in which reactive actions are insufficient.

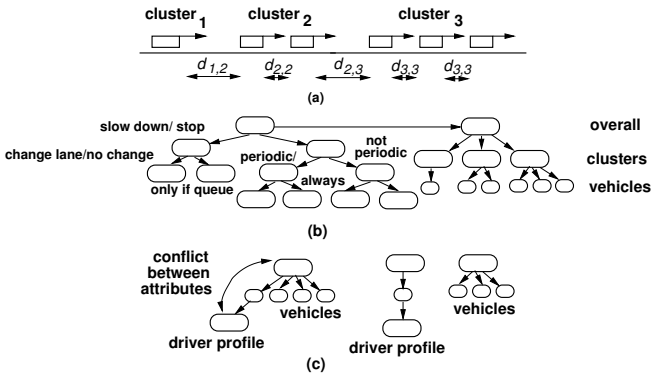


Fig. 1. Simple traffic scene

Example: Let's consider the simple traffic situation in Figure 1 to illustrate the three challenges in scene understanding. Figure 1(a) presents five vehicles moving on the road. Scene understanding must first identify the three vehicle clusters, where a cluster comprises of the vehicles moving according to the same pattern (e.g., similar speed and speed variations). This pattern must be different from the patterns of other clusters. If the clusters move with different speed then only speed is sufficient for cluster identification. However, if two clusters are moving at the same speed then additional attributes are needed for differentiating the clusters, such as the interspacing $d_{i,j}$ between the vehicles. A possible differentiation criterion is that interspacing is significantly larger than the average of the other interspacing.

An important aspect in concept identification (including finding concept attributes) is the identification of the necessary and sufficient information that makes the identification process possible. Moreover, inferring the information needed for scene understanding helps solving the ambiguities that can occur between different concepts with common attributes. Hence, concept identification relies not only on finding similarities between concepts but also outliers.

Another important objective of scene understanding is getting insight into the causes of the relations between concepts, and solving the ambiguities that occur during this step. These relations are not directly evident from the description of a traffic application. For example, there can be multiple causes for vehicle slow down, e.g., potholes, stopped cars, traffic lights, and flooded areas. However, these causes can be often distinguished from each other by using sufficient relevant attributes. For example, potholes force cars to mainly slow down and change lanes, while stopped vehicles cause vehicles only rarely to switch lanes or to stop. Moreover, traffic lights impose a periodic stopping of all cars, while for other periods cars movements are not affected. Finally, flooded areas cause all vehicles to stop and wait until the cars in front pass. In this case, there is no attribute that distinguishes the four cases. Instead, the ontological hierarchy in Figure 1(b) must be used for getting insight into the traffic scene and disambiguate the possible cause - effect relations by finding the most likely cause.

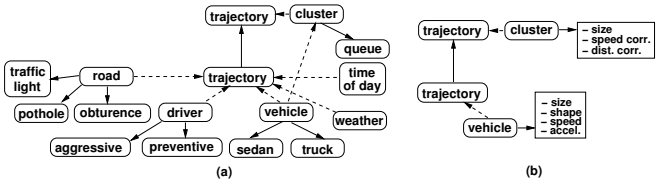


Fig. 2. Simple ontology for traffic applications

Once the concepts and their relations in a scene are understood, the information is used to understand the expected dynamics (evolution) of the scene and the emergence of new relations. For example, if the driver profile in Figure 1(c) does not match the speed attribute of the cluster, it is likely that that vehicle will leave the cluster in the near future.

III. Ontology Description

An ontology describes the concepts, attributes (properties), and permanent relations among the concepts of an application class. Ontologies offer an abstract yet complete description of the possible situations that can occur in reality. Each ontology defines the concepts (components) that form a real situation, the relations according to which the concepts are linked together, and the attributes (features) of the concepts, the constraints of the attribute values (e.g., sequencing over time, impact of events, etc.). The instantiation of an ontology for a specific scenario is useful to find the mathematical models that describe the scenario. The models result as a composition of the models describing the concepts and relations identified from the ontology.

Guarino and Welty [4], [5] propose that ontologies are characterized using metrics, like unity and identity. Unity states that all instances of a concept are linked to the concept through a well defined set of properties. Rigidity indicates that properties do not change within a time window but then can change as a result of an event. A rigid property carries identity condition if it the existence of the property implies that the involved instances are equal. We propose a similar approach based on the common attributes of the instances of a concept.

A. Traffic Scene-related Ontology

In our approach, every concept represents a group of instances that share a common set of attributes and are distinguishable from other instances and concepts by another set of attributes. Attributes are invariant features of instances. There can be various perspectives to describe the invariant character of attributes, such as invariant over time, space, population, etc.

The meaning of a traffic scene is defined in terms of a set of basic semantic elements, which cannot be defined using more basic elements and can be estimated based on the inputs coming from sensors. The basic semantic elements (BSEs) to be identified and analyzed include the following aspects:

- *Vehicle attributes*: Some of the typical vehicle attributes include kind, speed, acceleration, position, and trajectory.
- *Driver's driving profile*: A profile includes his/her preferred style of driving depending on traffic and weather

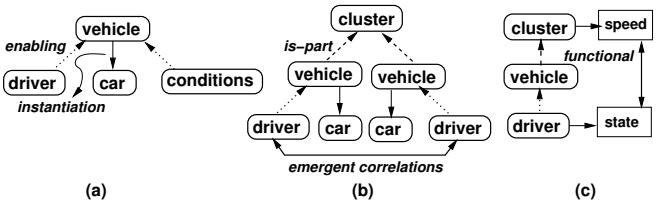


Fig. 3. Relations in ontologies for traffic applications

conditions. For example, the driver's profile describes the likelihood of changing the speed or trajectory (e.g., switching the lanes).

- *Clusters of vehicles*: Clusters are formed by vehicles that travel while having a common set of stationary attributes, such as a constant number of vehicles in the cluster and vehicle speed variations and inter-vehicle spacing that pertain to well-defined (yet unknown) ranges.
- *Cluster attributes*: Every cluster is characterized by attributes like size (number of vehicles), speed range, trajectory, time of formation and time of dispersion. Clusters have also attributes that are different from the attributes of vehicles, e.g., spacing between cars.
- *Cluster-level, social behavior*: The way in which the drivers forming a cluster change their driving behavior based on the cluster characteristics, e.g., drivers decide to adapt to the speed of the other drivers in the cluster, or start looking for opportunities to leave the cluster.
- *Cluster dynamics*: Vehicle clusters go through modifications, such as a cluster splitting into sub-clusters and different clusters merging into a single clusters. Another kind of interaction is if two clusters automatically correlate their attributes, like speed.
- *Road conditions*: This refers to special road conditions, e.g., the position of potholes, traffic signs, and stopped vehicles.
- *Weather conditions*: This aspect relates to the nature of weather conditions, such as the position of ice and water on the road.

The elements above define a simple ontology for traffic applications. They are the basic elements involved in traffic and are used for expressing the possible interactions and correlations in traffic scenes. Every particular traffic scene is a specific instance of the ontology. Understanding the behavior of traffic involves constructing the traffic scene corresponding to the ontology.

Figure 3 illustrates the nature of relations between the concepts of a traffic-related ontology. Figure 3(a) shows concept instantiation and enabling relations. Instantiation, indicated with solid line, defines that concept *car* is a more specific concept than concept *vehicle*. Some of the defining attributes of concept *vehicle* have a more constrained description for *car*. For example, attribute *size* is restricted to a smaller range. Still, the constrained attribute allows distinguishing the concept from other concepts instantiated based on concept *vehicle*. Enabling relations, shown with dotted lines, indicates that the characteristics of the related class are used to control (refine)

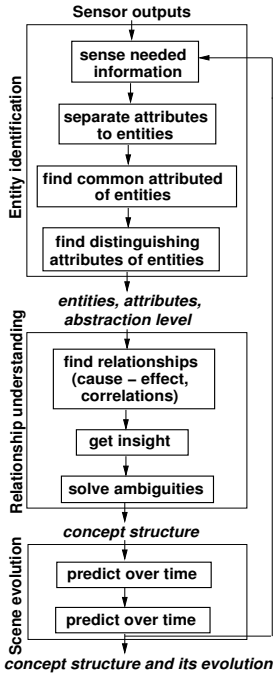


Fig. 4. Traffic scene understanding methodology

more specific attribute values for the concept. For example, the attributes of concepts *driver* and *conditions* restrict the attribute values of concept *vehicle*.

The *is-part* relation in Figure 3(b) (shown with dotted line) defines that all attributes of the target concept depend on attributes of the originating concepts or instances. This means that for every attribute of the target concept, every participating concept has at least one attribute that influences the attribute. For example, all attributes of concept *cluster* (of vehicles) depend on the attributes of the instances (vehicles) that form the cluster. Note that the attributes of the target concept might depend also on other attributes than those of the originating concepts or instances. A concept set C_{in} is a complete description of the $is-part$ relation with concept C, if there is no other concept that has a $is-part$ relation with concept C.

B. Constructing Traffic Scene Representations

Figure 4 presents the proposed methodology to construct representations of traffic scenes. The methodology has three steps: (i) identifying the entities that participate to the scene, such as individual vehicles, vehicle categories (i.e. sedan, truck, SUV, etc.), vehicle clusters, road obstacles, traffic lights, and so on; (ii) understanding the relations between the found entities, and (iii) predicting the dynamics of the scene based on analytical models for the scene as well as sensed data acquired in real time.

The first step, entity identification, finds the participating entities based on their distinguishing attributes (which separate them from other entities). Every instance, i.e. vehicle in a scene, has physical attributes (PAs), some of which can be measured directly through sensors, e.g., position, dimension, weight, temperature, time, and so on. Some of the sensor

readings might be unavailable at a certain moment. Every attribute can have values from a (constrained) domain. In addition, attribute descriptions might include constraints defined over the associated attributes, including constraints between the goal of the application and attributes. Common attributes of multiple instances enable the identification of the categories to which the individual entities pertain to, like the vehicles that form a cluster.

Example: For traffic applications, possible PAs are position (of a vehicle), time, dimension and weight (of a vehicle). Speed, another attribute of a vehicle, is represented as the following tuple: $(\frac{x-x_0}{t-t_0}, (x,t), (x_0,t_0), t > t_0, t - t_0 < \epsilon)$. The speed attribute (the first component of the tuple) is defined using two other associations of PAs, (x,t) and (x_0,t_0). Besides, the constraints $t > t_0, t - t_0 < \epsilon$ must be valid to compute correctly the speed attribute. Similarly, the interspacing between two vehicles A and B is defined as $(x_A - x_B, (x_A,t), (y_B,t), x_A > x_B)$.

The second step of the methodology finds the causal relations among concepts, such as the reasons that produce certain constraints and patterns of the attributes of entities. Causes that are directly observable through sensors are utilized to formulate hypothesis on the causal relations that might originate the constraints [16], [20]. Other potential causes, which are not directly observed, are formulated based on the ontology of traffic scenes during the insight getting step. The likelihood of (observable and unobservable) causes are computed using Bayesian networks, a popular causal reasoning procedure [15].

The third step constructs the analytical models starting from the identified scene elements and the causality relations between them. The analytical models include the mathematical expressions that characterize the attributes of the elements as well as the expressions of the causal relations. These models are then used to predict the future dynamics of the represented scene. Section IV illustrates the algorithm to predict traffic scene evolution, including the behavior of the current clusters and their merging and splitting.

IV. Analytical Descriptions using Ontologies

The identified scene description is utilized to produce automatically analytical descriptions of scene. The analytical models serve to predict the future behavior and characteristics of a real scene. Such predictions are important to make decisions on how to optimize the architecture and parameters of the monitored traffic system, including traffic lights and feedback given to the participating drivers.

Figure 5 presents the procedure to estimate the dynamics of a traffic situation that is monitored through sensors. The goal is to predict all events within a time window of adjustable the length. Events correspond to cluster merging and splitting. The window length is modified at run time such that the predictions are sufficiently close to the events of the real life traffic scenario.

The procedure computes a stochastic sequence of events knowing that every cluster merging is followed by one or more

```
(1) while (events are possible) {
(2)  event = select next event;
(3)  time = expected time of event;
(4)  S = set of clusters that could participate to event;
(5)  for (all consecutive pairs i,j in set S) {
(6)    for (all speed vi of cluster i) {
(7)      for (all speed vj of cluster j) {
(8)        if (vi < vj) {
(9)          p = probability of merging Cvi and Cvj;
(10)         if (p > pthresh) {
(11)           T = time of merging event of Cvi and Cvj;
(12)           add new splitting events at time Tj and with probability p;
(13)           for (all speed vii < vi)
(14)             Tii = time of splitting Cvi for speed vii;
(15)             add new event at time Tii and with probability p;
(16)           }
(17)           for (all speed vjj < vj) {
(18)             Tjj = time of splitting Cvj for speed vjj;
(19)             add new event at time Tjj and with probability p;
(20)           }
(21)         }
(22)       }
(23)     }
(24)   }
(25) }
(26)}
```

Fig. 5. Predicting traffic dynamics

cluster splittings, so that the created vehicle groups reflect the desired driving behavior of the participating vehicles. In an optimal situation, no vehicle should drive at a lower or higher speed than the one desired. The algorithm considers every cluster pair i and j and estimates the likelihood of the clusters being merged. Then, using analytical equations defined for clusters in the ontology, the algorithm computes the merging time (instruction 11). The condition of a subsequent cluster splitting is also verified next (instruction 12). The possible splitting times corresponding to different splitting scenarios are evaluated in instructions 13-16 and 17-20. The splitting times are also computed using analytical equations stored in the ontological structure.

Stochastic events are generated at steps 9, 12, and 19 to indicate cluster mergings and splittings with a probability p which can be estimated based on the attributes of the instances (e.g., vehicles) forming a cluster. Hence, the output of the procedure is a set of possible event sequences and the likelihood of each sequence. Only the sequences with likelihood above a threshold limit are returned.

The next subsection illustrates the analytical equations that are used in estimating the expected time moments and probabilities of cluster merging and splitting events.

A. Case Study

The prediction procedure is illustrated for a traffic flow characterization application. The traffic flow consists of a sequence of vehicle clusters, such that the vehicles in a cluster move with similar speed. The ontology description specifies that all a cluster is defined by all vehicles of similar speed (speed is a common attribute) and located within a close neighborhood within each other. The distinguishing of a cluster from another cluster is based on differentiating attributes, such as different vehicle speeds or inter-vehicle distances that are

beyond the specified limit. Hence, assuming that $\epsilon_{cluster} \geq 0$ is the accuracy of the modeling, the speed of any pair of vehicles i and j of a cluster is related by the following constraint:

$$|v_i - v_j| \leq \epsilon_{cluster} \tag{1}$$

The parameter $\epsilon_{cluster}$ is an input to the analytical model. Depending on the road conditions, the minimum size of a cluster is defined as Lim. For very narrow roads, Lim can be one. The value increases for wider roads.

At time moment t, every cluster C_I is characterized by a set of attributes, including the number of vehicles in the cluster $N_{C_I}(t)$, the physical size of the cluster, e.g., the length $L_{C_I}(t)$ measured on the x coordinate, and the characteristics of the vehicles forming the clusters. These attributes are Physical Attributes (PAs) and sensed directly through sensors.

Without limiting the generality of the prediction method, the model assumes that the vehicle characteristics refer mainly to the driving style of the driver, such as the most likely speed v at which it would drive, if no speed constraints are set by the other cars participating to the traffic. Different formalisms have been proposed for modeling the driving behavior, including Markov Processes, neural networks, and Bayesian networks. Because of the constraining due to the other drivers of a cluster, the memory aspect is less important. Therefore, the driving behavior is expressed as a set of bins, each bin representing the most likely driving speed, if no constraints exist. If B bins are used for modeling, then bin $k, k \geq 1$, corresponds to the speed range $[v_{min}+(k-1)\ \frac{v_{max}-v_{min}}{B}, v_{min}+k\ \frac{v_{max}-v_{min}}{B}]$, where v_{min} and v_{max} are the minimum and maximum speed of a vehicle in traffic. $N(C_I, k, t)$ is the number of vehicles in cluster C_I and which pertain to bin k at time moment t. The specific procedure to model the driver behavior is also indicated in the ontology.

Cluster descriptions in the ontology include two rules that (i) express the behavior of a cluster, and (ii) capture the interaction between pairs of clusters. Regarding the first rule type, depending on its composition, a cluster can split into two subclusters, if a group of vehicles starts moving faster than the rest of the cluster. A cluster is considered to be split, if the distance between the two groups is larger than the limit D_{Lim}. Also, a vehicle i attempts to exit a cluster if its desired speed differs from the cluster's speed by more than ϵ_{split}:

$$|v_i - v_{cluster}| \geq \epsilon_{split} \tag{2}$$

The desired speed is estimated based on the predicted driver's profile.

Using the mathematical constraints defining the conditions of a cluster split, the prediction procedure estimates the probability of having a split. Assuming that all vehicles attempt to reach their desired speed (predicted based on the expected driver behavior), the probability of splitting cluster C_I into two sub-clusters $C_{I,1}$ and $C_{I,2}$ is as follows:

$$p(C_I, C_{I,1}, C_{I,2}) = \frac{N_{C_{I,1}}}{N_{C_I}} \tag{3}$$

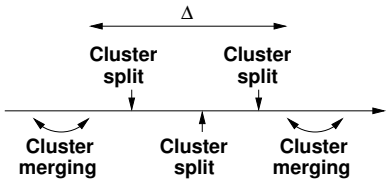


Fig. 6. Predicting traffic dynamics

where $N_{C_{I,1}}$ is the number of vehicles that have a desired speed larger than v_j, where j is the first vehicle of the cluster meeting condition (2). The speed v_j becomes the speed of the new cluster $C_{I,1}$, while the vehicles of cluster $C_{I,2}$ continue to move at the same speed as cluster C_I.

The time at which the separation occurs is modeled as:

$$t_{split,C_I,C_{I_1},C_{I_2}} = \frac{D_{Lim}}{v_j - v_{C_I}} + t_r \tag{4}$$

where t_r is a random variable modeling the decision of a vehicle to start moving faster than the rest of the cluster. It decreases with the value of $p(C_I, C_{I,1}, C_{I,2})$:

$$t_r = \frac{1}{p(C_I, C_{I,1}, C_{I,2})} t_{r0} \tag{5}$$

t_{r0} is a constant. These analytical expressions are stored for the rule describing cluster splitting.

With respect to the second rule type, two clusters interact with each other only if the second cluster moves at a speed slower than the first cluster. The interaction between the clusters can proceed along two situations: (i) the second cluster decides to slow down to the same speed as the first cluster, and (ii) the second cluster (or parts of it, if there was a split) catches up with the slower, first cluster and the two merge. The result of the first situation (no interaction) is that the two clusters have the same speed. For the second situation, the time of merging is expressed as:

$$t_{merging,C_1,C_2} = \frac{D_{1,2}}{|v_{C,2} - v_{C,1}|} \tag{6}$$

where $D_{1,2}$ is the distance between the two clusters, and $v_{C,2}$ is the speed of cluster C_2, which is the faster cluster that succeeds cluster C_1. $v_{C,1}$ is the speed of cluster C_1.

The prediction model uses the analytical equations of the identified ontological entities to express the clusters' behavior as a stochastic sequence of rules, where the above rules are applied depending on specific conditions, like the traffic situation and the driving patterns of the drivers. The model offers statistical predictions indicating the likelihood of having cluster mergings and splits. Figure 6 illustrates the method.

Between successive cluster interactions, i.e. by applying the rules for merging and no interactions, clusters are partitioned into sub-clusters by applying zero or more times the rule for cluster splitting. The time between two consecutive interactions is defined as:

$$T = \frac{\Delta}{|v_{fastest,C_2} - v_{slowest,C_1}|} \tag{7}$$

where Δ is the distance between the two clusters. $v_{fastest,C_2}$ is the speed of the fastest sub-group that splits from the

second cluster C_2 and $v_{slowest,C_1}$ is the speed of the slowest sub-group that splits from the first cluster C_1. The modeling method must estimate $v_{slowest}$ and $v_{fastest}$ as well as the expectations of the splittings that are likely during the time T.

The probability of having a certain speed v as $v_{fastest}$ is estimated as follows. It assumes that all splittings during time T produce only sub-clusters that have speed v as the highest. There are numerous scenarios that result in this case. For example, there can be a single split that generates one cluster of speed v while the rest keeps the previous speed. Or, two splits, in which first a lower speed is produced followed by a second split that generates the desired speed. In general, for k splits, the first $k-1$ produce sub-clusters of lesser speed than v, while the last partition generates the sub-cluster of speed v. Assuming that the number of expected split points is $E[\#_{splits}]$ then the probability of having a sequence of splits such that speed v is highest is expressed as:

$$p(v) = p(C_I, C_v, C_{v_c})(1 - E[\#_{splits}] \sum_{\forall v_i < v} p(C_I, C_{v_i}, C_{v_c})) \tag{8}$$

The expected number of split points $E[\#_{splits}]$ depends on the time interval between consecutive splittings, as defined by formula (4). Computing a value for the expectation is difficult as the time intervals depend on the order of splittings. Instead, an approximation cab be calculated using the following bounds:

$$\frac{D_{12}\ \Delta v_{min}}{D + t_r \Delta v_{min}} \leq E[\#_{splits}] \leq \frac{D_{12} \Delta v_{max}}{D + t_r \Delta v_{max}} \tag{9}$$

The expression results by observing that relationship (4) is monotonically increasing with the value of the speed v, thus the minimum time interval between splitting results for maximum difference in speed between v and the speed of the cluster.

V. Scene Element Identification using Sensed Data

In the implementation of the methodology, sound based localization is used to extract the intra-cluster vehicular distance. Sound localization is the process of identifying the spatial coordinates of a sound source based on the sound signals received by a microphone array. Other features of the sound sources, such as spatial coordinates, distance from other sound sources, speed of movement etc. are derived from these localization estimates. This information is utilized for identifying the elements of a traffic scene, including vehicle clusters, cluster merging and cluster splitting.

Each sensor node is equipped with a pair of microphones in order to perform time difference of arrival (TDOA) estimation [1]. The maximum likelihood technique is applied on the Generalized cross-correlation (GCC) equation to identify the Angle of arrival or Direction of arrival (DoA) of the sound source [6].

The elements of the traffic scene are identified using data clustering. Data clustering groups the data into clusters based on a criterion function. The SVM-based clustering algorithm is unsupervised and does not assume any prior knowledge of the input classes [21]. The algorithm starts by running a binary SVM classifier against a dataset with randomly labeled input vectors. This first step is repeated until convergence is acheived and the acceptable number of KKT violations are encountered. This is accomplished by using different number of allowed violators until a lower bound is reached such that the SVM converges.

After the first step is over, the confidence parameters for classification are available. The data with the lowest confidence has the worst mislabeled vectors. So, the labels for this data are switched to the other class. The SVM is again run on this dataset and therefore has a higher probability of convergence and results in fewer mislabeled vectors. The process is repeated until there is no further improvement in results.

The following 6 scenarios are simulated to identify some of the semantics of the traffic. The experimental setup for each of the scenarios is shown in Figures 7a to 7d. Two sensing nodes (N1 and N2), which are PSoC1 embedded processors from Cypress Semiconductors Corporation, are used for sound localization. The distance between the two nodes, D, is 72 inches. Four different positions (P1 to P4) are considered to simulate the movement of vehicles. Experiments are done with 1, 2, and 3 sound sources to simulate different cluster size for different scenarios. Sound samples are captured using the microphones and the localization estimates are used to extract different attributes such as spatial coordinates of the sound sources, their speed of movement etc. First, the feature vectors are clustered into different classes using SVM based clustering and then the clustered data is used to train the SVM classifier.

1) Single vehicle in favorable driving conditions: In this scenario, only one vehicle is tracked (Figure 7a). By favorable driving conditions, we mean that there is no sudden change in the speed of the vehicle from one position to the other, that is, it remains constant.
2) Cluster of vehicles in favorable driving conditions: A cluster of vehicle is simulated using multiple sound sources, one for each vehicle in the cluster (Figure 7b). The favorable driving conditions are simulated as described previously.
3) Single vehicle in bad driving conditions: This scenario is simulated as described in Figure 7a but in this case, the speed of the vehicle varies at different positions. In real environment, the variation of speed may be due to potholes, bad weather or road conditions. Even with varying speed, the classifier can identify the class of unknown feature vectors based on the intra-cluster object distance.
4) Cluster of vehicles in bad driving conditions: A cluster of vehicles in bad driving conditions is simulated using multiple sound sources (Figure 7b). In this case, the speed of the cluster is changing with time but the speed of all the vehicles within the cluster is the same. Also, the intra-cluster vehicular distance remains same at different sampling positions.
5) A vehicle joining a cluster of vehicles: Figure 7c de-

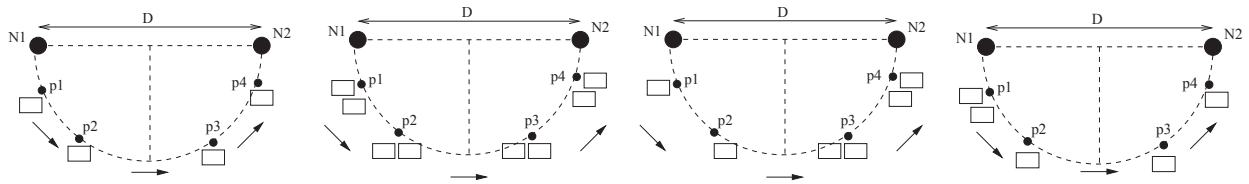


Fig. 7. Simulation of (a) Single vehicle, (b) and (c) Cluster of vehicles, (d) Vehicle joining a cluster

TABLE I
SPATIAL COORDINATES CORRESPONDING TO DOA ESTIMATES

		Position 1		Position 2		Position 3		Position 4	
		X	Y	X	Y	X	Y	X	Y
CS1	V1	4.07	52.68	14.30	67.77	34.14	60.32	54.25	49.53
CS2	V1	6.14	47.14	14.26	67.58	38.41	59.35	55.88	47.04
	V2	22.44	60.31	26.26	72.89	52.31	57.47	69.92	39.87
CS3	V1	4.37	47.87	15.08	66.85	35.12	63.08	55.88	47.04
	V2	12.40	58.76	22.73	66.32	42.78	57.58	64.43	41.19
	V3	21.19	61.84	27.60	74.16	51.05	53.97	70.31	40.09

scribes the scenario of a vehicle joining a cluster. A single vehicle is present at positions $p1$ and $p2$ and a new vehicle joins it at position p3and creates a cluster of size 2. This is simulated using a single sound source at p1and $p2$ and two sound sources at positions $p3$ and $p4$. Experiments are also performed with different cluster sizes.

6) A vehicle splitting from a cluster of vehicles: Figure 7d describes the scenario where a vehicle splits from the cluster. This is simulated using two sound sources at position $p1$ representing two vehicles and a single sound source at positions $p2$, $p3$, and $p4$.

The DoA estimates are used to compute the spatial X and Y coordinates of the vehicles. The results are shown in Table I. The dimensions are in inches.

Certain factors affect the localization accuracy of the nodes. For example, as the distance between the microphone pair and the sound source decreases, the DoA estimates become coarser. Physical parameters such as speaker width and sensitivity of the microphone contribute towards measurement errors. Accuracy of experimental setup and error due to elevation of microphone and sound source cause results to deviate from the actual measurements.

Even though these factors skew the results, the clustering accuracy for a data set with 8 feature vectors used for the experiment was found to be 87.5%. The classification accuracy in identifying the 6 scenarios is 100%. The SVM-based clustering and classification algorithms were executed on the PC (Intel Core2Duo, 1.3GHz, 1GB RAM) and PSoC 5 (ARM Cortex core,80MHz) and the respective execution times for the learning step are 3 ms and 13 ms.

VI. CONCLUSIONS

This paper proposes a methodology to model the dynamics of traffic scenes, including the participating vehicles, vehicle clusters, attributes and relations of all scene elements, and related events, like cluster merging and splitting. The main steps of the methodology find the elements of a scene, identify the relations among the elements, and construct analytical prediction models for the traffic scene dynamics. Compared to other methods, this methodology constructs the models online using data from embedded sensors.

REFERENCES

[1] A. Umbarkar, V. Subramanian, A. Doboli, "Low-Cost Sound-based Localization using Programmable Mixed-Signal Systems-on-Chip", *Microelectronics Journal*, Vol. 42, No. 2, February 2011, pp. 382-395.

[2] M. Baykal-Gursoy, Z. Duan, H. Xu, "Stochastic Models of Traffic Flow Interrupted by Incidents", *Control in Transportation Systems*, 2010.

[3] D. Belomestny, V. Jentsch, M. Schreckenberg, "Completion and continuation of nonlinear traffic time series: a probabilistic approach", *Journal of Physics A: Math. Gen.*, 36, 1136911383, 2001.

[4] N. Guarino, C. Welty, "Unity and Individuality: Towards a Formal Toolkit for Ontological Analysis", *Proc. European Conference on AI*, 2000.

[5] N. Guarino, C. Welty, "Towards a Methodology for Ontology based Model Engineering", *Proc. ECOOP Workshop on Model Eng.*, 2000.

[6] D. Halupka, J. Mathai, P. Aarabi, Sheikholeslami, *A. Robust sound localization in 0.18 um cmos*, IEEE Trans. Signal Processing, 53(6), 2005.

[7] T. Hastie, R. Tibshirani, and J. Friedman, "Elements of Statistical Learning", *Springer*, 2001.

[8] D. Helbing, A. Hennecke and M. Treiber, "Phase diagram of traffic states in the presence of inhomogeneities", *Phys. Rev. Lett.*, 82, 4360, 1999.

[9] D. Helbing, "A section-based queuing-theoretical traffic model for congestion and travel time analysis in Networks", *J.Phys.A.*, 36, L593, 2003.

[10] D. Jost, K. Nagel, Probabilistic Traffic Flow Breakdown In Stochastic Car Following Models, *Traffic and Granular Flow*, 2003.

[11] S. Krauss, "Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics", *PhD Thesis*, University of Cologne, Germany, 1997. www.zpr.uni-koeln.de.

[12] S. Lammer, R. Donner, D. Helbing, "Anticipative control of switched queueing systems", *Euro. Physics J. B*, 63, 341-37, 2008.

[13] M. J. Lighthill, G. B. Whitham, "On kinematic waves: II. A theory of traffic on long crowded roads", *Proc. Roy. Soc. London Ser. A*, 229, 317-345, 1955.

[14] K. Nagel, M. Schreckenberg, "A cellular automaton model for freeway traffic", *Journal de Physique I*, 2 (12), Dec. 1992, pp.2221-2229.

[15] R. Neapolitan, "Learning Bayesian Networks", Prentice Hall, 2004.

[16] J. Peral, "Causality. Models, Reasoning and Inference", *Cambridge University Press*, 2000.

[17] I. Prigogine, F. C. Andrews, "Boltzman-like approach for traffic flow", *Operations Research*, 8(6), 789-797, 1960.

[18] P. I. Richards, "Shock waves on the highway", *Operations Research*, 4, 42-51, 1956.

[19] T. Singliar, M. Hauskrecht, "Modeling Highway Traffic Volumes", *Proceedings of the 18th European conference on Machine Learning*, 2007.

[20] P. Spirtes, C. Glymour, R. Scheines, "Causation, Prediction, and Search", *The MIT Press*, 2000.

[21] S. Winters-Hilt, S. Merat, *SVM clustering*, BMC Bioinformatics, 8(Suppl 7): S18., 2007.

Notes

Notes

Towards a Model-Driven Engineering Software Development Framework

Maxime Perrotin, Julien Delange and Samir Bennani
European Space Agency, TEC-SWE
Keplerlaan 1
2201AZ Noordwijk, The Netherlands
Email: firstname.lastname@esa.int

***Abstract*—Design and Implementation of Safety-Critical Systems is becoming very difficult becauses it involves many requirements coming from different engineering domains. Due to the increase of complexity, software of such systems can no longer be produced with traditional methods, which show their limit over time. In that context, new development approaches have to be introduced to avoid actual development traps and pitfalls. Among them, the Model-Driven Engineering approach consists at representing system artifacts with models and auto-generate the code by refining them from high-level concepts down to the code. However, as for every new approach, it also brings new problems such as requirements consistency among the different notations (models) as well as integration issues (for example, making sure that implementation code from different models will behave correctly when merged on a single execution platform).**

This article presents our experience for integrating Guidance and Navigation Control (GNC) algorithms designed with Application Models (Simulink) with Architecture Models (AADL). The process relies on code generator for both models and integrate it on a typical execution platform. In particular, we focus on the challenges of the integration, illustrating the practical problems we faced for producing a space system using a Model-Driven Engineering Approach.

***Keywords*-AADL, TASTE, Simulink, MDE**

I. Introduction

A. Context

Safety-critical systems are getting more complex, collocating more functions on the same computing platform. As a consequence, their design becomes more complicated, leading to a long and potentially painful design process. Designers have to take into account requirements coming from different domains and specified with heterogeneous formalisms.

For that reason, producing the system using traditional methods is no longer feasible : checking impact between all requirements disseminated across different specifications and notations is impossible, especially when the process is not automated. For that reason, new approaches must be designed. In our context, the Model-Driven Engineering (MDE) approach aims at separating system concerns in models, let engineers focus on their part of the system while tools automate the integration and ensure consistencies between modelling artifacts.

B. Problem

One key aspect of the MDE methodology is the separation of concerns: each engineer focuses on designing and implementing his part of the system dedicated to his domain while specific tools process each implementation artifact, ensure their integration and preserve a semantic consistency between notations.

Despite having a clear separation between each domain, several problems remain when implementing a full MDE approach. First of all, because of the use of different notations, some requirements are sometimes captured twice in different models using heterogeneous notations. Then, tools are expected to ensure their consistency but engineers are also requested not to break them when manipulating models. Another issue is about simulation and implementation correctness: even when a system was intensively tested using simulation functions, its integration as an implementation code can generate a lot of errors. Most of the time, this is due to the heterogeneous nature of the execution platform, whose environment is different from the simulation. Thus, having a dedicated process that ensures a smooth integration of system functions by enforcing their behavior correctness is a must.

In that context, to improve MDE approaches, it is worthwhile to identify all traps and pitfalls in order to strengthen the overall approach. Outcome of such investigations would make tools more resilient to potential integration errors, the type of issue which is typically discovered just before completion of a project, when few resources are available

C. Outline

The remainder of this paper is structured in two main sections. The first one is an overview of the tools we use: the TASTE toolset (Architecture modelling [1]), and Simulink [2] (Software modelling) and their integration. Second part of this paper presents our latest experiments to design a spacecraft system using our MDE toolset by designing GNC algorithms on top of a distributed architecture designed with TASTE. We provide a feedback about traditional traps and pitfalls of such integration, leading to an open discussion about potential improvements of the whole MDE development process.

II. Background

A. Simulink

Simulink consists of a graphical modelling language and a set of tools for designing software (an example is shown in figure 2). It focuses on the definition of functional concerns and is mostly used to design algorithms in specific engineering domain (power control, navigation, etc.).

Simulink is a well-known and established tool, providing convenient notation to abstract engineering concepts with a user-friendly simulation interface. Thus, it is a very efficient tool for engineers to prototype, design and implement their part of the system. In the context of the space industry, it is used in many engineering domains, from mechanical to robotics. In the present case-study, it was used to produce the GNC algorithms of a launcher.

B. TASTE

TASTE [3] is a project developed, maintained and supported by the European Space Agency. It aims at providing a MDE toolset for the production of safety-critical systems. For that purpose, it defines the system using three views:

1) The **Data View** (DaV) specifies data types and encoding functions used to communicate between system components using the ASN.1 [4] language (see listing 1 for an example). It corresponds to the external interfaces of the system, as specified in the ICD that specified all interfaces between the sub-systems (with their types, properties, etc.).
2) The **Interface View** (IV) enumerates system functions (what the system is doing), their requirements and constraints (timing, data protection, etc.) and interactions among then (communication channels) using the AADL [5] language. Function connections reference the **Data View** to specify types being used. As this model remains descriptive (a graphical sample is shown in figure 3), it has to be associated with code that implements functions.
3) **Deployment View** (DeV) defines the execution platform (processors, buses, etc.) of the system, its configuration as well as deployment of functions (from the **Interface View**) into it. It also uses the AADL [5] for that purpose, an example is shown in figure 4.

These models are processed with their functional code to automatically produce system implementation (as shown in figure 1) through the following steps:

- The **Data View** is translated into declarations (data types, functions) in the target code (C, Ada, etc.) so that we can use the same interfaces with different languages.
- The **Interface** and **Deployment Views** are processed to create an architecture code that supports the execution of the functional code. This aims at creating and configuring resources (tasks, mutexes, etc.) that reflect requirements specified in the interface view (period, deadline, etc.).
- The **Integration Process** compiles the functional code from the user with the architecture code generated from models, producing the program to be deployed on the execution target. This code is also automatically tailored to the target operating system. As for now, our toolchain supports several architecture (x86, SPARC, etc.) and different Operating System for safety-critical systems such as Linux or RTEMS [6].

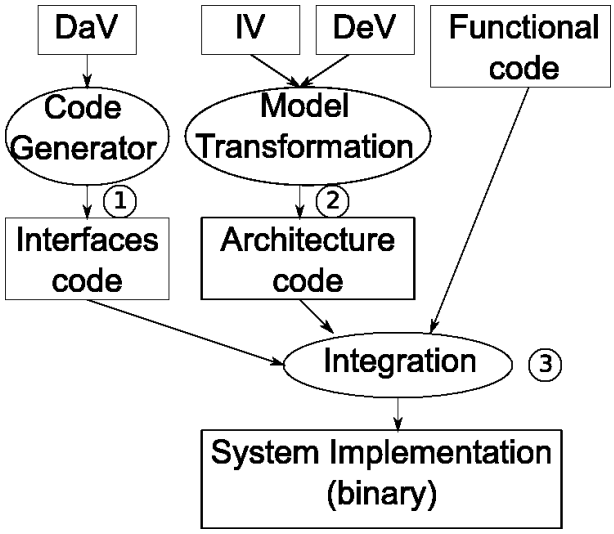


FIGURE 1: *TASTE development process*

C. Integration of Software Models into Architecture Models

Our process (figure 1) automatically deploys application code (written by domain-specific users, such as electrical/mechanical engineers) on top of architecture code. The former can be designed using either regular (C, Ada) or modelling (Simulink, SDL, etc.) languages. Our toolchain automatically generates glue code that connects functional blocks, enabling communication between code written with different languages and executed on heterogeneous architectures.

However, when integrating application models (such as Simulink), algorithms requirements must comply with the architecture constraints (timing requirements of the **Interface View**, interfaces definition of the **Data View**, etc.). Using a traditional, manual integration, no check is performed, lack of compliance between models is discovered either after integration (at best), or during execution (at worst). The following section contains the description of a complete case study we conducted in order to experiment and validate the use of a model-based approach with the TASTE tools.

III. Case-Study and feedback

A. Overview

In order to validate our approach and tools, we have built a system to support the validation of the navigation algorithms

(GNC) of a onboard launcher software. In that context, we have developed (or reused) a large set of Simulink models that represent the environment of the software: the sensors and actuators on one side, and the flight dynamics on the other side. This way we have the means to generate realistic data at runtime to feed the control laws and run in closed loop.

On the other side, we have the flight code of the control laws in Ada language. The challenge is to put both pieces together, and make them run on different platform: first natively on Linux to test the integration, and later on a mixed platforms (x86/Linux for the environment models, and Sparc/Leon for the control law running in real-time). At runtime, we want to observe data and plot the control laws' main parameters. TASTE provide the means to achieve these goals.

B. Application modelling

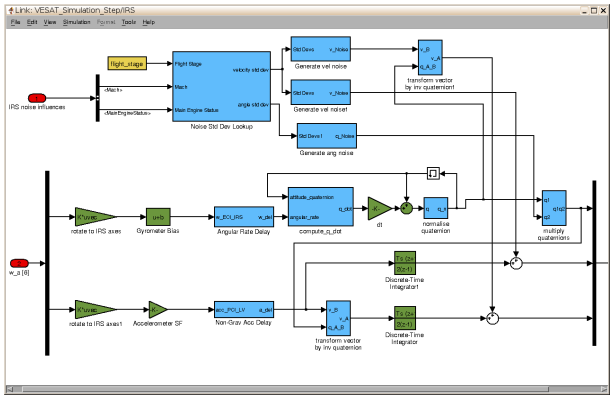


FIGURE 2: *Simulink model of our case-study*

Modelling the application with TASTE was done in three steps

1) specify the data types in ASN.1 to describe the messages exchanged between our functions (listing 1)
2) capture the logical architecture of the system (figure 3) into an Interface View that references the Simulink model (shown in fig:simulink-model).
3) model the deployment of the application, by mapping the functions onto hardware components, and connect them with buses (figure 4).

From these models, TASTE generate *skeletons*, that consist in empty code blocks that would contain the application (Simulink models). Once these code blocks have been filled by the user, tools automatically create all the glue code that is necessary to implement the system (communication, etc.) without having to manually tweak the interfaces. This auto-connects blocks from different implementation languages (Ada, C, Simulink, etc.) smoothly, without having to change the communication mechanisms. Using these tools gave us guarantee that there would be no inconsistencies in data representation for each block. This allows to start running simulations very quickly and make rapid prototyping of embedded application without having to tweak application code.

```
VEGA DEFINITIONS ::=
BEGIN
T-GNC-LV-SIM-CONTEXT ::= SEQUENCE {
    attitude-quaternion             T-QUAT-FLOAT32,
    ng-vel-incr-irs                 T-VECT3-FLOAT32,
    ng-vel-incr-accelero            T-VECT3-FLOAT32,
    filtered-angles-sample-1        T-VECT3-FLOAT32,
    filtered-angles-sample-2        T-VECT3-FLOAT32
}

T-GNC-LV-SIM-INPUTS ::= SEQUENCE {
    sequ-exec-request-vect
        T-HAS-SEQUENCE-EXEC-BEEN-REQUESTED-VECT,
    tvc-set-point-eng-vect
        T-TVC-SET-POINT-ENG-VECT,
    racs-ev-cmd-vect
        T-RACS-EV-CMD-VECT
}

T-QUAT-FLOAT32 ::= SEQUENCE
        (SIZE(size-T-QUAT-COMPONENTS)) OF T-FLOAT32
#  ...
END
```

Listing 1. Data View of our case-study

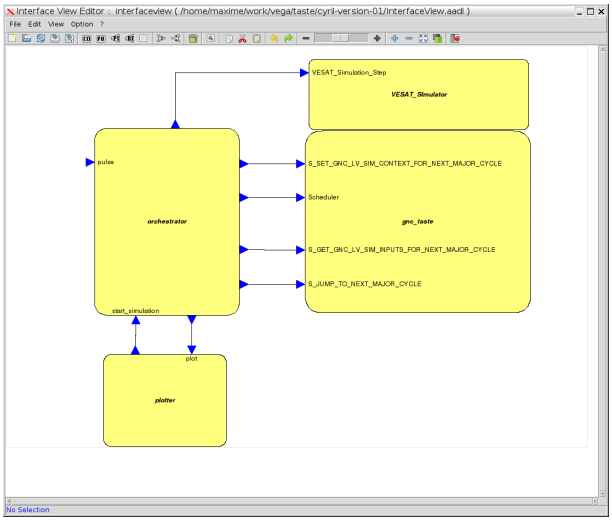


FIGURE 3: *Interface View of our case-study*

C. Feedback

The experience gives us the ability to produce a large scale launcher simulator that we are now running to cross-check the control laws of our launchers. Until now, we did most of these simulations entirely within the Matlab/Simulink environment. It is also a very powerful and effective approach; the drawback is that the execution is not representative of real targets, and that the integration of the code later on in the real onboard software can become much more difficult when done at a later stage. Using automated tools to make the models and code integration from the very early stages

of the development brings significant added value and makes things simpler to integrate for non-software people.

However, we experienced two major issues when integrating Simulink code on the real target. First of all, the generated code from application models is not fully consistent with the models specification, leading to errors that were not seen during simulation. This brings us to debug the generated code by hand to discover the problem. This should be automatically detected by the tools. Also, another issue was the programming skills of engineers : the toolchain required adaptation of existing Simulink models to the TASTE interface. This job requires some model refactoring to fit TASTE and Simulink models and was not easy for non computer-scientist engineers. As a result, we spent a lot of time explaining the design process to engineers so that they can use our toolchain.

IV. Conclusion

This article presents our feedback about the issues of the deployment of software models with architecture, particularly regarding potential errors that are introduced during the integration phase, at the latest phases of system production. These experiments were done in the context of internal projects at the European Space Agency, while integrating GNC algorithms (designed and tested usign simulation functions from Simulink) with an execution runtime.

These issues convince us to strenghthen the overall development process and propose new functionalities that aim at checking system compliance between execution and simulation. For example, being able to monitor system interfaces and check correctness between simulation and execution would help developers but also support the overall development process, providing artifacts required for system validation.

A. Perspectives

Verification between models could also be introduced earlier in the development process, prior to system integration. For example, it would be possible to check compliance between heterogeneous models before generating or integrating code. This engineering effort, even if technically feasible, would require a static analysis of the model, which requires a huge maintainance effort due to the number of application languages supported by our toolchain.

sectionAknowledgment

The authors would like to thank the VEGA project and the TEC-ECN and TEC-SW laboratories for their support and contributions to this paper.

Acronyms

ASN	Abstract Syntax Notation
ICD	Interface Control Description
GNC	Guidance and Navigation Control
MDE	Model-Driven Engineering
TASTE	The ASSERT Set of Tools for Engineering

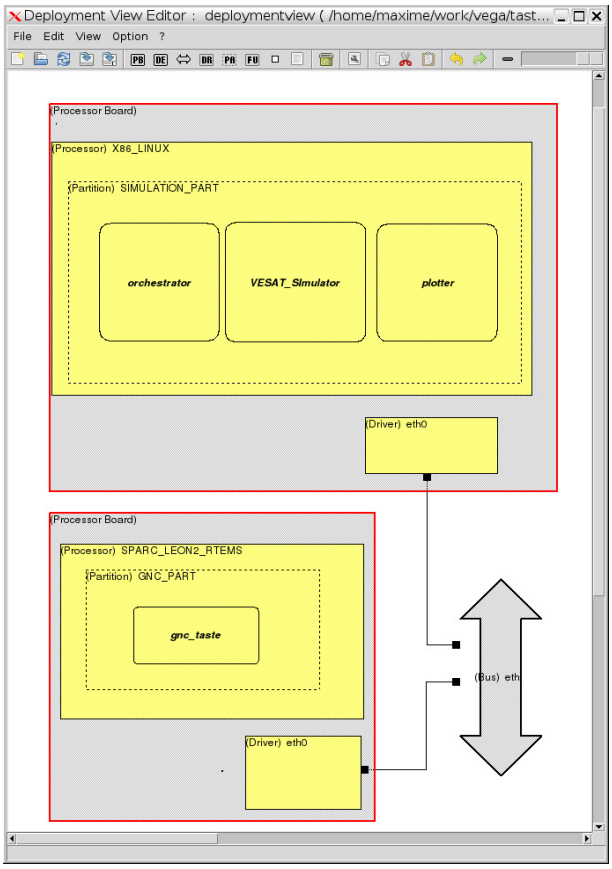


FIGURE 4: *Deployment View of our case-study*

References

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, ser. SEI Series in Software Engineering. Addison-Wesley, 2003.

[2] O. Beucher, *MATLAB und Simulink (Scientific Computing)*. Pearson Studium, 08 2006.

[3] M. Perrotin, E. Conquet, J. Delange, T. Tsiodras, and A. Schiele, "TASTE A real-time software engineering tool-chain," in *Proceedings of the 15th International Conference on System Design Languages*, 07 2011.

[4] O. Dubuisson, *ASN.1 Communication entre systèmes hétérogènes: Guide d'utilisation, manuel de référence*, ser. Collection Technique Et Scientifique Des Telecommunications. Springer, 1999.

[5] As-2 Embedded Computing Systems Committee SAE, "Architecture Analysis & Design Language (AADL)," SAE Standards n^o AS5506, November 2004.

[6] "RTEMS : Real-Time Operating System for Multiprocessor Systems," http://www.rtems.com.

Notes

Notes

Integration of Mixed-Criticality Cyber-Physical Systems with Criticality Layers

Dionisio de Niz
SEI - Carnegie Mellon University
Pittsburgh, PA, U.S.A.
dionisio@sei.cmu.edu

Anthony Rowe
ECE - Carnegie Mellon University
Pittsburgh, PA, U.S.A.
agr@ece.cmu.edu

***Abstract*—Large Cyber-Physical Systems such as avionics and automotive systems often require large integration efforts between third-party components. These components provide functionality at different levels of criticality yet share many of the same underlying resources (CPU, Memory, Network, Disk, Transducers). As a result, protections mechanisms are needed to prevent lower-criticality task from interfering with higher-criticality ones. In this paper, we discuss how traditional temporal protection mechanisms such ARIC 653 partitions fail to fully protect high-criticality tasks from lower-criticality ones. We then show how the Zero-Slack QRAM scheduler (ZS-QRAM) can be used in multi-layer systems to avoid these problems. Furthermore, we propose the use of criticality layers based on the asymmetric protection scheme of ZS-QRAM in order to simplify this integration, increase its robustness, and reduce its resource usage. Finally, we discuss some open issues that need to be addressed in order to remove some of the limitations of this approach.**

I. Introduction

In modern Cyber-Physical Systems (CPS), such as avionics and automotive systems, there is increasing pressure to reduce cost while increasing functionality and managing physical resources like power and heat. This has resulted in the consolidation of functionality of different components into shared hardware resources (e.g. processor and memory). Unfortunately, such sharing can lead to interference across tasks (e.g. one task using the processor longer than expected) each of which may have different criticality requirements for the system. For instance, if we deploy an ABS braking system task on the same processor as a navigation system task of the car, if the latter does not release the processor on time it could make the former miss its deadline. This problem is a concern of growing interest and has spawned initiatives to investigate building blocks that can be used to safely construct mixed-criticality systems like the Mixed-Criticality Architecture Requirements (MCAR) [1] as well as standards like ARINC-653 [2] and ISO-26262 [3].

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The consolidation of mixed-criticality systems demands the temporal protection of critical functionality. Traditional approaches such as ARINC 653 offer a symmetric protection scheme where not only higher-criticality functionality is protected from lower-criticality one but also lower-criticality is protected from higher-criticality one. Unfortunately, this type of protection can lead to criticality inversion, i.e., lower-criticality tasks blocking a higher-criticality one. Specifically, symmetric protection can stop a higher-criticality task if it tries to run longer that its allocated budget in order to allow a lower-criticality task to run. In contrasts, Zero-Slack Rate-Monotonic (ZSRM) scheduling [4] provides an asymmetric protection scheme where a lower-criticality task can be stopped in order to ensure that higher-criticality tasks meet their deadline but a lower-criticality task cannot interfere with higher-criticality ones. The Zero-Slack QoS Resource Allocation Model-based (ZS-QRAM) [5] scheduler extends this concept even further to systems where tasks can have the same level of criticality but provide different utility to the user. In this case, when an overload occurs, the lower-utility tasks are degraded to a lower execution rate (longer periodicity) to allow higher utility tasks to keep executing at a high rate.

We believe that asymmetric temporal protection should be the norm when integrating mixed-criticality CPS. This type of protection enables the creation of criticality layers that support layered certification standards like DO-178C [6] and ISO-26262 [3]. In this paper, we present a layered architecture based on the asymmetric temporal protection of ZS-QRAM and related mechanisms to support different aspects of the applications. Finally, we discuss the issues that still need to be addressed to provide a comprehensive solution.

II. ZS-QRAM

In this section, we provide background information related to ZSRM and ZS-QRAM as a basis for our discussion on using it in a layered manner. ZSRM is a fixed-priority preemptive scheduling approach for uniprocessors. In ZSRM, a *criticality* value (ζ_i) is associated with each task τ_i to reflect the task's importance to the safety of the CPS[1].

[1]Our convention is to use lower values to indicate higher criticality.

These tasks are periodic with a period T_i, an implicit deadline at the end of the period, and two execution times. The first execution time is considered the WCET during a nominal mode of operation, called *Nominal-Case Execution Time* (C_i), and the other is called the *Overloaded-Case Execution Time* (C_i^o) that is considered the WCET during an overload situation (e.g. when the number of objects to avoid is unusually large). ZSRM works on top of traditional priority-based preemptive real-time schedulers. It is based on the observation that criticality inversion only matters under overload conditions. We use this observation to create two execution zones for each task τ_i. In the first zone, every task is allowed to execute as normal, while in the second zone, every task $\tau_j|\zeta_j > \zeta_i$ is suspended. This suspension effectively blocks the interference of lower-criticality tasks in the case of an overload condition, up to the completion of the task activation. It must be noted that τ_i itself can also be suspended by a task $\tau_c|\zeta_c < \zeta_i$ in τ_c's second zone. The execution zones partition the execution of each task into two modes: the normal mode (*N mode*) and the critical mode (*C mode*). Our scheduling algorithm then calculates the last instant at which a task can suspend lower-criticality tasks in order to finish before its deadline. This instant is known as the zero-slack instant (Z_i) and is used at runtime to setup a timer when a job $J_{i,k}$ from τ_i arrives. If such an timer expires, then the lower-criticality tasks are suspended, but if $J_{i,k}$ finishes before the timer expires, then no suspension is performed and the lower-criticality tasks are allowed to continue.

ZS-QRAM builds upon ZSRM by adopting mechanisms from the Quality-of-Service (QoS) Resource Allocation Model (Q-RAM) [7]. Q-RAM uses utility functions that describe the different QoS levels that tasks can obtain along with the resources (e.g., processor time) that they consume and the utility that the user derives from each QoS level. In simple configurations, Q-RAM primarily takes advantage of the fact that as applications (e.g. video streaming) increase their QoS level, the incremental utility to the user decreases. This is known as *diminishing returns*. For instance, in a video streaming application increasing the frames per second from 15 to 20 gives the user higher utility (i.e., perceived quality) than increasing from 20 to 25 frames per second. Q-RAM uses the utility functions to perform a near-optimal allocation of resources to different tasks exploiting the diminishing returns property. In particular, the diminishing returns property manifests itself in these functions as a monotonically-decreasing utility-to-resource ratio. This ratio is know as *marginal utility*. Q-RAM uses the marginal utility to perform the allocation one increment at a time starting with the increment that derives the largest utility for the smallest allocation of resources (largest marginal utility). In each of the subsequent steps, it selects the next largest marginal utility increment until the entire resource (e.g. CPU utilization) has been allocated. In the ideal case, the marginal utility of the last allocation (QoS level) of all the tasks is the same.

ZS-QRAM is designed for tasks whose different QoS levels are implemented using different task periods. Task periods are mapped to allocation points in the utility functions. ZS-QRAM first considers utility functions based on the C_i of the tasks, and utilizes Q-RAM to do an initial allocation where each increment in the allocation is represented by an increasingly shorter period. If a task overloads at runtime, an overload management mechanism is used to degrade tasks (by selecting a longer period) to keep the taskset schedulable. This mechanism uses task utility functions based on their C_i^o to select the tasks that render the least utility per unit of CPU utilization (marginal utility).

ZS-QRAM supports an admission test that uses two steps. First, it performs an initial Q-RAM allocation using the nominal marginal utility of the tasks. In other words, it builds utility functions assuming tasks run for their C_i and follows the Q-RAM allocation order until the available CPU utilization is fully allocated. Second, once the Q-RAM allocation is completed, the ZS instant Z_i of each task τ_i is calculated. This instant is then used to degrade tasks τ_j with lower utility than τ_i. To support this degradation, tasks are structured as a set of task modes with different utility and the degradation is configured as a mode transition.

III. ZS-QRAM Layer Hierarchy

ZS-QRAM embeds both the criticality-based and the utility-based scheduling and enforcement policies. Since the criticality-based policy has precedence over the utility-based one, they form layers where tasks scheduled based on criticality have precedence over tasks scheduled based on utility. We identify these layers as the *criticality layer* and the *utility layer* respectively.

The criticality layer is divided into multiple sub-layers (called simply layers when appropriate) based on criticality levels (used in the taskset). Criticality layers provide two important properties that simplifies the integration of mixed-criticality systems: *layer isolation*, and *layer overbooking*. Layer isolation guarantees that a higher-criticality layer cannot be affected by a lower-criticality layer. In other words, every task τ_i in a high-criticality layer are guarantee to execute for C_i^o before its deadline whether or not lower-criticality layers exist. This implies that these lower-criticality layers can be added on top of higher-criticality ones without compromising their guarantees. On the other hand, layer overbooking allows layers of different criticality to share CPU cycles increasing the system capacity to fit more features. In this case, the lower criticality layer uses these shared cycles (meeting its deadline) if no task τ_j in the higher criticality layer exceeds its nominal execution time C_j. This overbooking allows an efficient and safe resource sharing across criticality layers. It is worth noting that both

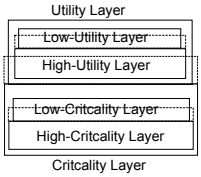


Figure 1. ZS-QRAM Layer Hierarchy

the layer isolation and the layer overbooking applies to both the criticality and the utility layers.

The utility layer is also divided into multiple utility sub-layers. However, in this case, each sub-layer is composed of the set of task modes that are active at the sub-layer's marginal utility level. That is, each sub-layer represents a QoS adjustment (implemented as a period degradation) at each level of overload. Both criticality and utility layers are depicted in Figure 1

It is worth noting that defense systems' features are generally classified as safety-critical and mission-critical. This classification matches the semantics of the criticality and utility layers of ZS-QRAM. Similarly, within the safety-critical class, some features are more critical than others demanding a different degree of certification suing standards like the DO-178-B/C [6]. In particular, DO-178 describe five *Design Assurance Levels* (A-E) that are assigned to different features depending on the consequence of their failure. This corresponds to how stringent a validation and verification process must be applied. From a timing guarantee point-of-view, Vestal [8] observed that this relates to the pessimism (safety-margin in DO-178 parlance) of the worst-case execution time of the task(s) that execute the particular feature. The layer isolation offered by ZS-QRAM allows each layered to be evaluated disregarding any low-criticality layer. Each layer needs to be considered at its own safety margin and WCET (C_i^o). As lower-criticality layers are verified and validated, the safety margins are reduced, reducing in turn the WCET of higher-criticality layers (each task in these layers is assume to run for C_i only) while the current layer is assumed to run at its proper criticality margin (each task τ_j in this layer is verified at C_j^o).

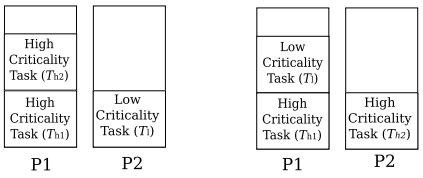
The utility layer extends the Design Assurance Level concept beyond safety (that matches criticality) into mission value. In this case, the adaptation is encoded into a task (or feature) adaptation where the task is not dropped but instead adapts to the available resources according to its mission value.

A. Resource Sharing in a Layered System

Sharing resources in our layered architecture must be properly supported to avoid violating layer guarantees. In particular, in order to share a resource (e.g. shared data structures) across tasks, some concurrency control needs to be used. In these cases, locking protocols have been designed for general-purpose computing, and for real-time systems. In particular, the real-time locking protocols (e.g. priority inheritance protocols [9]) are aimed at minimizing the time that a low priority task can prevent a high-priority task from executing, a.k.a. *priority inversion*. In mixed-criticality systems two problems must be resolved. First, we need to minimize not only the priority inversions, but also the criticality inversions that can occur when resources are shared across criticality layers. The Priority-and-Criticality Inheritance Protocols [10] (PCIP) were designed for this purpose. These protocols ensure that a low-criticality task that holds a lock requested by a high-criticality task cannot be stopped by any medium-criticality zero-slack enforcement. The PCIP protocols also provide a schedulability test that takes into consideration the wost-case blocking times observed under these protocols.

Beyond schedulability, the job-stopping that happens in the criticality layers can leave resources locked indefinitely (if the lock owner is killed). As a result, when resource sharing is used, priority demotion should be used instead of job dropping. In this case, the priority of the target job is demoted to a non-real-time priority and allowed to continue on cycles not used by the real-time tasks. Additionally, this strategy must be complemented with a deadline-miss notification (say a signal handler) that warns the job of the deadline miss so that it can take the appropriate actions like releasing locks.

B. Criticality Layers in Multi-Core Processors



(a) Crit. Inv. (b) No Crit. Inv.

Figure 2. Criticality Inversion in Allocation

Multi-core processors bring new challenges to providing isolation layers. In general, real-time scheduling for multi-core processors can be divided into global scheduling, partitioned scheduling and semi-partitioned. In global scheduling, tasks are allowed to execute on any available core. In partitioned scheduling, a task is allocated to a core at boot time and is restricted to only execute on that processor. Finally, in semi-partitioned scheduling, most tasks are restricted to a single core while others are partitioned into two segments with one running on one core and the other on another core. In general, global scheduling can reach higher schedulable utilization when combined with some proportional fairness policy, i.e., ensuring that each task constantly receives a portion of the processing cycles equal to their utilization. However, global scheduling tend to be costly due to the cost of migrating a task from one processor to another (e.g. moving the cache content). In contrast, partitioned scheduling tends to be more practical and simple to use,

but it may suffer from fragmentation cost due to unused utilization of a processor (where no single task can fit). However, in practice this fragmentation is rare and for safety critical systems, the simplicity of a partitioned approach is highly desirable. For this reason, we propose the use of partition scheduling for layered mixed-criticality systems.

In partitioned scheduling of mixed-criticality systems, the criticality inversion problem can arise at the task-allocation level. A given allocation could favor a low-criticality task at the expense of a high-criticality one. For instance, consider three tasks τ_{h1}, τ_{h2}, and τ_l of high, high, and low criticality respectively, each having a normal utilization $\frac{C_i}{T_i}$ of 40% . Assuming we only have two processors P_1 and P_2, and τ_{h1} is already deployed on P_1, and the three tasks do not fit on P_1 together, then we are forced to pack either τ_{h2} or τ_l on P_1 and the other to P_2. Packing τ_{h1} and τ_{h2} together, and τ_l by itself is a possible task allocation decision (see Figure 2(a)). In fact such an allocation decision is commonly used in legacy systems that try to isolate criticality levels. However, observe that such task allocation leads to a criticality inversion problem. In this scenario, if all the tasks overload, τ_{h2} may miss its deadline but τ_l will not i.e. our allocation decision protected τ_l (a low criticality task) at the expense of τ_{h2} (a high criticality task). Conversely, deploying τ_{h1} and τ_l together and τ_{h2} by itself removes this criticality inversion (see Figure 2(b)). Note that using a criticality-aware uniprocessor scheduling algorithm such as ZSRM will ensure that τ_l cannot steal cycles from τ_{h1} within processor P_1 under overload scenarios.

In [11] we developed a mixed-criticality task allocation algorithm called *Compress-on-Overload Packing* (COP) that first allocates the highest-criticality tasks as if they were running for its overloaded execution time C_i^o. Once the processors are full, it then recalculates the allocated utilization of the processors as if the tasks were now running for their nominal execution time C_i. Finally, the lower-criticality tasks are packed "on top" of the previously allocated tasks. It is worth noting that this scheme allocates the tasks in criticality layers, from the highest to the lowest criticality. This packing highlights the advantage of sharing resources between low and high criticality tasks.

IV. Open Issues

Asymmetric protection for mixed-criticality systems needs to be extended across other subsystems of the computing platform to act as a comprehensive solution. Areas currently not covered include: the Input-Output subsystems including interrupts, DMA, and lower-level sensors and actuations. In distributed systems, protection mechanisms for network bandwidth would also be required. In the multicore arena, resources shared across cores such as cache and memory banks require new mechanisms for asymmetric temporal protection. Finally, the mixed-criticality synchronization protocols need to be extended for multiprocessors (and multicores).

V. Concluding Remarks

In this paper, we proposed the use of the asymmetric temporal protection of ZS-QRAM to integrated mixed-criticality CPS in a layered architecture. We presented the different mechanisms that ZS-QRAM uses to build a hierarchy of criticality layers and the advantages of this hierarchy for the certification of these systems in the light of safety standards like the DO-178C. In addition, we presented the mechanisms to support resource sharing within and across criticality layers. We discussed the criticality inversion problem in partitioned multiprocessor scheduling and our allocation algorithm designed to minimize it. Finally, we discussed the open issues that need to be addressed in order to provide a comprehensive solution. In the end, we believe that the layered architecture presented here and the supporting mechanisms provide a strong design platform for mixed-criticality systems that is well aligned with the philosophy of the certification authorities facing commercial and military CPS.

References

[1] D. Homan, "Designing future systems for airworthiness certification," http://www.cse.wustl.edu/~cdgill/CPSWEEK09_MCAR/, 2009.

[2] *Avionics Application Software Standard Interface*, ARINC Std. 653p1-2, 2005.

[3] ISO, "Road vehicles – functional safety," http://www.iso.org/iso/catalogue_detail.htm?csnumber=43464.

[4] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *RTSS 2009*, dec. 2009.

[5] D. de Niz, L. Wrage, N. Storer, A. Rowe, and R. R. Rajkumar, "On resource overbooking in an unmanned aerial vehicle," in *ICCPS*, 2012.

[6] S. C. of RTCA, "DO-178C, software considerations in airborne systems and equipment certification," 2011.

[7] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for qos management," in *RTSS*, dec 1997.

[8] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," *In Proceedings of RTSS*, pp. 239–243, 2007.

[9] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *Computers, IEEE Transactions on*, vol. 39, no. 9, Sep 1990.

[10] K. Lakshmanan, D. de Niz, and R. R. Rajkumar, "Mixed-criticality task synchronization in zero-slack scheduling," in *Proceedings of the IEEE RTAS*, 2011.

[11] K. Lakshmanan, D. de Niz, R. R. Rajkumar, and G. Moreno, "Resource allocation in distributed mixed-criticality cyber-physical systems," *ICDCS*, 2010.

Notes

Notes

Analytic Certification Technologies for Military Avionics

Russell B. Kegley and Jonathan D. Preston
Lockheed Martin Aeronautics Company
Fort Worth, Texas, USA
{russell.b.kegley, jonathan.d.preston}@lmco.com

***Abstract*—Historic approaches to upgrading the capabilities of military aircraft by inserting new technologies have become so costly that warfighters may be forced to operate with less than current technology can deliver. Even inserting new upgrades alongside the legacy systems can be very expensive if it requires large modifications to legacy software, triggering extensive retest. A way to meet this challenge is to insert upgrades alongside legacy systems, virtually replacing old capabilities with new while leaving the old software in place. This approach carries with it new challenges which might be met with analytic innovations.**

Keywords-component; incremental upgrade, embedded avionics, timing, specification, reuse, verification, validation, complexity

I. INTRODUCTION

Increasingly, the lifetimes of military aircraft platforms are exceeding 40 years. Modernization of the basic airframe in the form of better engines or strengthened components is a well-understood practice. However, the industry must find new paths for upgrading software-provided capabilities. Past approaches accomplished mission software and hardware upgrades by essentially removing the old components and replacing them with entirely new systems inserted into the existing airframe. Budget realities going forward make this an unaffordable model, making it essential to be able to continue to use some functions from the legacy system while inserting new or updated capabilities alongside them. Accomplishing these incremental system upgrades challenges the analytical frameworks that defense contractors have relied on to date to ensure predictable, bounded integration times and costs, and to generate performance and deadline guarantees required for system acceptance. In this position paper, we outline the problem and point to specific challenges which we believe are key technologies required to meet them.

II. MISSION AVIONICS UPGRADES: THE PAST

Past approaches to mission avionics upgrades have followed a few general approaches. Figure 1 shows a typical legacy mission avionics system, in which the primary software domains (Controls, Displays, Weapons, etc.) are distributed over a set of computing nodes; typically the nodes have nearly identical processing technology, with some differences to accommodate specialized I/O. The domains usually have well-defined boundaries which often correspond to hardware boundaries, meaning that changes can be localized fairly narrowly. The paper by Seeling [1] gives a fairly rare published detailed view of a specific avionics system in the context of reconfiguration options; most such system descriptions are not readily available.

Over a period of 7-15 years after an aircraft has been introduced, the industry develops new technologies and enhanced capabilities which start appearing in new products, but which are not available in the older aircraft. During this time, military customers create a body of experience and knowledge of how to use the aircraft in actual missions, and also discover functions which, if added to the aircraft, would make it better-suited to their needs. The defense purchasing community and the aircraft manufacturers jointly identify a set of capabilities to be introduced, and create a procurement program which involves removing the existing mission avionics system and replacing it *in toto* with a newly-developed set of hardware and software. While this new system almost always contains the same capabilities as the old software, these

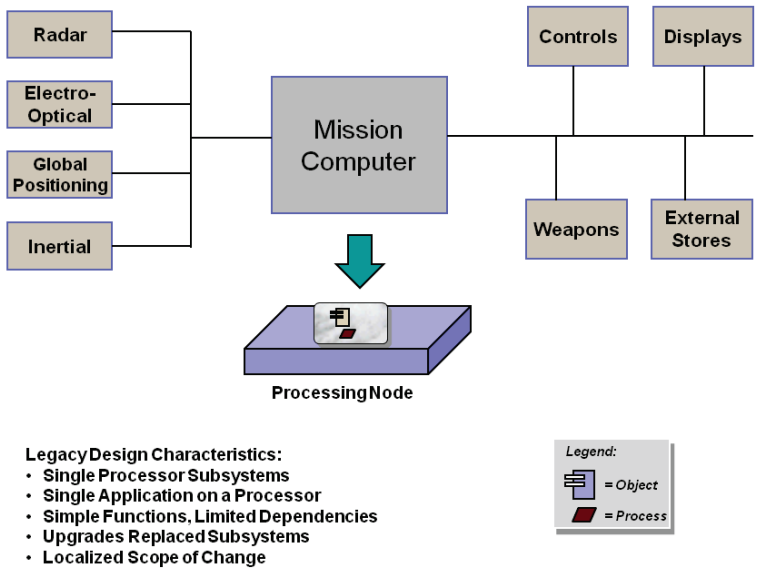


Figure 1. Legacy missions avionics system

functions may be expressed in new forms, and of course the new system has additional functions which were not present in the system. Since the end product is a complete replacement, there are few backward compatibility issues to be handled. The new capabilities are typically well-crafted into the user interface, and in most respects the integration is not much harder than a complete new product would be.

Procurement bodies and contractors have long recognized that this is not the most desirable scenario, and have proposed others. The overriding concern, as noted by Duren [2] is to minimize changes to existing software, since this incurs the greatest expense. One approach proposed by Luke et al., RePLACE, uses emulation to rehost legacy software unchanged on a newer, faster processor [3]. While this method lets the integrator continue to use old software without change, and provides the CPU power to support new functions, it was not primarily concerned with the issues involved in making the new work with the old. An alternative to executing the legacy software unchanged is to simply port it in a functionally-unchanged form to the new environment; an extreme example of this is discussed in Murray [4], where assembly language code for an obsolete processor is ported to the new system. It is hard to imagine this scenario being affordable today.

The primary obstacle to continuing to do wholesale system upgrades is that while it is easier in some ways, complete mission hardware and software replacements are very expensive; forecasts for defense spending seem to rule out most of these upgrades in the future. The high cost also has another, even more important downfall: when capability upgrades are delayed for years or perhaps decades because of budget constraints, warfighters are faced with the prospect of having to conduct their missions with less capability than the technology would allow.

III. Mission Avionics Upgrades: A Possible Future

A great deal of the cost associated with the upgrade practice just discussed is the result of completely replacing all of the hardware and software systems. In many cases, the old functions may have been sufficient even with new capability insertions, but the replacement approach rewrites that software so it can be hosted on the new hardware. The cost escalates further in that these "new" software functions, which actually simply replace existing functions which were still adequate, must be exhaustively tested.

Performing future upgrades could be made more affordable by carefully matching current capabilities against the desired capabilities and finding ways to keep those functions in the current system which still meet the new mission requirements. Leaving much of the current system in place and simply adding new software capabilities and hardware could eliminate a large part of the upgrade cost. Figure 2 shows this scenario in the context of a proposed refresh of a radar subsystem; this is a fairly common replacement as radar hardware and a software technology advance, and represents a great increase in warfighter capability. As shown, updating the radar involves accommodating the new temporal architecture the subsystem uses, meaning that some "glue code" needs to be written to bridge this difference. Since the radar brings new capabilities which replace earlier functions dependent on the old radar subsystem, a new sequence of function calls or object method invocations is created on the inserted system which renders the earlier function void. In this proposed upgrade scenario, the old radar mission thread software is not removed from the legacy software; it is simply no longer exercised by any stimulus, rendering it virtually removed.

The new capabilities are of course hosted on an updated computing node, which may have a different operating system and middleware, perhaps reused from a newer military aircraft

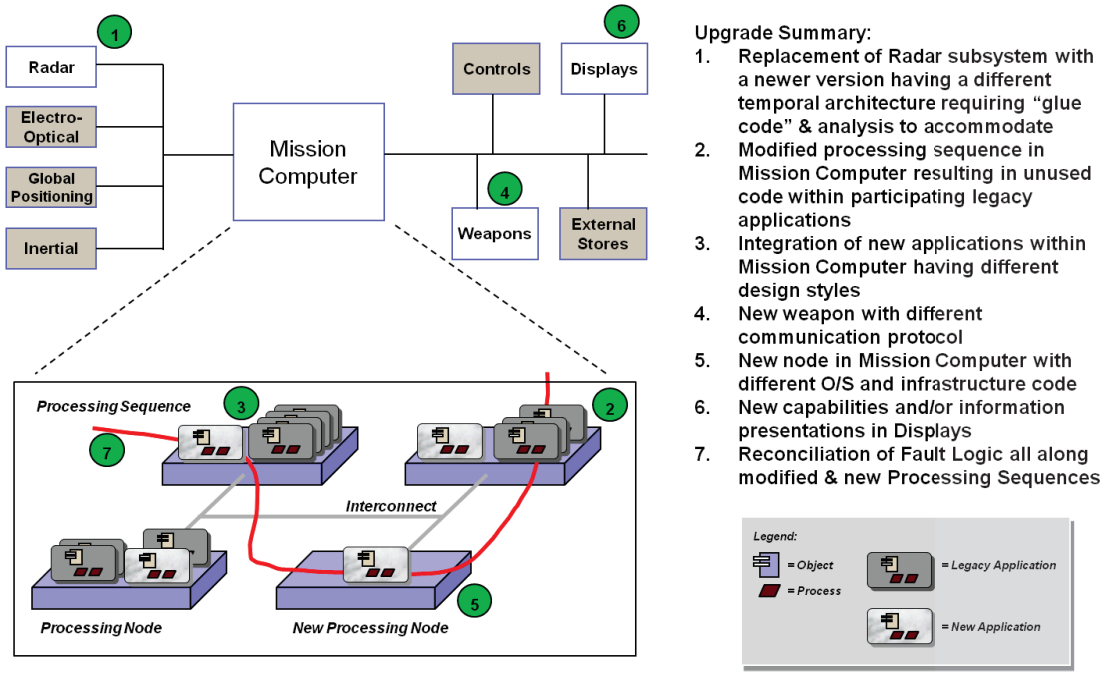


Figure 2. Proposed more affordable upgrade practice

program. This introduces the need to analytically demonstrate the harmony of fault logic and handling across the now-diverse architecture.

IV. Key Challenge Areas

Moving from the costly historical upgrade practices to a future practice which leaves still-functional software in place, and which virtually removes unneeded code instead of modifying the source, presents several challenges, which we detail below.

Temporal Architecture Differences The temporal architectures of the legacy and new functions may be vastly different. This difference could be relatively easy to bridge, like an old cyclic execution rate of 50Hz vs. a rate of 75Hz in the new software. However, the new software could be based on an event-driven paradigm which assumes immediate transmission of messages with an implied wait for response, such as a remote procedure call. This kind of difference may require more careful analysis, or cause changes to some of the cyclic rates in the old software to support the new software's deadlines. Thus, in some cases the new function might be able to be recast into the temporal vocabulary of the legacy behaviors, but when the "new" functions are, in fact, components originally developed for some other system, it might be required to use the new functions as-is and somehow harmonize the different temporal architectures.

Unused Code The idea of leaving some legacy code in-place while virtually turning off its execution is a novel one for military systems. Current standards for safety- or mission-critical applications call for physically removing all code that is not actually executed; a key enabler to an affordable incremental upgrade approach is some kind of virtual "dead code" removal, which can certify that the code cannot be exercised in the upgraded system while still leaving it in place in the legacy portion of the aircraft.

Application Architecture and Design Styles New applications likely employ different structuring approaches than legacy applications stemming from differences in programming languages, differences in computing resource constraints, and provisions for long term maintainability. Such differences come into play when it is desired to re-use legacy application code in part, or integrate software components from different suppliers. Reconciling differences can result in significant complexity to "glue code" written to integrate old and new components.

Communication Protocol Differences Applications typically communicate across the avionics architecture using messages. However, there can be great variety in the types and styles of communication and handshaking approaches used across different generations of applications. Different schemes include: Asynchronous unacknowledged messages, acknowledged delivery, remote procedure call (RPS), singlecast and multicast. Client/server and subscription based approaches have also been used.

Infrastructure Code and Support Services Applications from different eras and suppliers often have different infrastructure software requirements and assumptions, including those related to: operating system, security requirements, support libraries, load and startup, and persistent data storage, and scheduling.

Difference in Fault Tolerance Designs Generally, there are fault tolerance and system reliability requirements levied on avionic systems. Incremental upgrades need to be mindful of potential differences in fault detection, isolation, and response schemes. Often, fault tolerant designs are based on assumptions established very early in the initial design. Sometimes, documentation of requirements and underlying assumptions is lacking, and this can introduce risk as new functionality is added. Additionally, fault tolerance designs generally require full regression testing, even when the scope of intended design change is small and well known.

Documentation Coverage and Quality The behaviors of the legacy system may be incompletely documented or understood; in particular, the system's capabilities from a functional perspective may be so entwined that it is difficult to find ways to virtually sever the legacy system into behaviors that will continue to be exercised in the upgraded system and those that will be replaced by newly inserted technology.

V. Conclusions

We present a potentially more affordable route for inserting technology upgrades into military aircraft. While this approach avoids replacing existing, still-functional software, it carries with it new challenges that require analytical methods yet to be developed and proven. As Table I shows, the gap between legacy hardware, software infrastructure, and application designs can be large.

TABLE I. Historic vs. Future Upgrade Summary

	Legacy System	Future System
Processing hardware	Single CPU per subsystem Military-specific instruction set Single-core CPU's	Multiprocessor subsystems Commercial CPU's (suitably hardened) Multicore technology
Operating systems	Custom developed Fixed rate scheduling Timesliced temporal architecture	Commercial high assurance RTOS Mix of rate- and event-based processes Preemptive priority based scheduling
Networks	Custom or military standard	Commercial off-the-shelf
Applications	High degree of hardware dependency Minimal dependencies between applications Functionally constrained by computing resources, high complexity solutions	Hardware independent Increased interdependencies between software functions More complex capabilities by leveraging increased hardware resources

Upgrade scenarios which attempt to maintain large portions of the existing software in an untouched state while virtually replacing their capabilities presents multiple specific problems related to identifying and limiting the scope of changes, demonstrating the virtual excision of legacy functions without source code changes, and accommodating new computing architectures side-by-side.

REFERENCES

[1] Seeling, K., "Reconfiguration in an integrated avionics design," 15th Digital Avionics Systems Conference, 1996, pp.471-478.

[2] Duren, R., "Options for upgrading legacy avionics systems," 21st Digital Avionics Systems Conference, 2002, pp. 12-21.

[3] Luke, J., Bittorie, J.W., Cannon, W.J., Haldeman, D.G., "Replacement strategy for aging avionics computers," IEEE Aerospace and Electronic Systems Magazine, vol.14, no.3, Mar 1999, pp.7-11.

[4] Murray, J.P., Cole, C.K., Warlick, E.S., "Re-hosting the operational flight program for a tactical fighter plane's radar data processor: a software upgrade methodology," Proceedings of the IEEE 1996 National Aerospace and Electronics Conference, vol.2, pp.490-496.

Notes

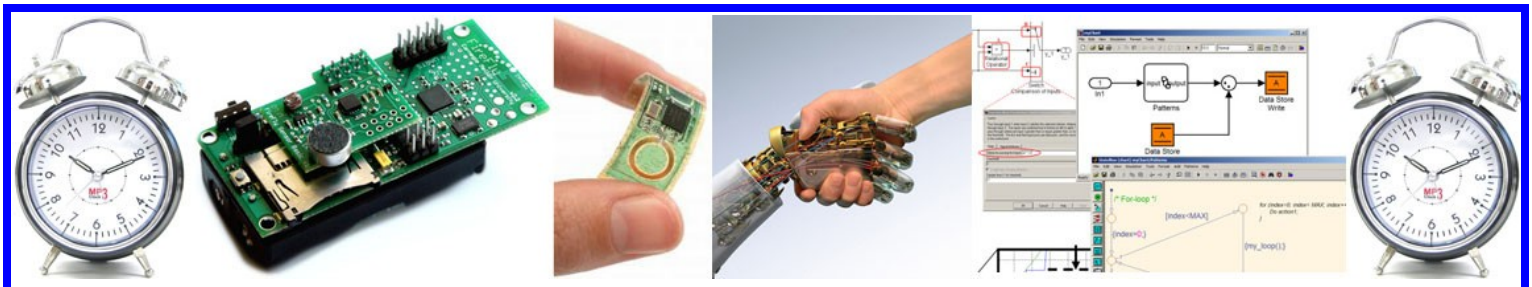
Notes

Notes

Notes

Notes

Notes



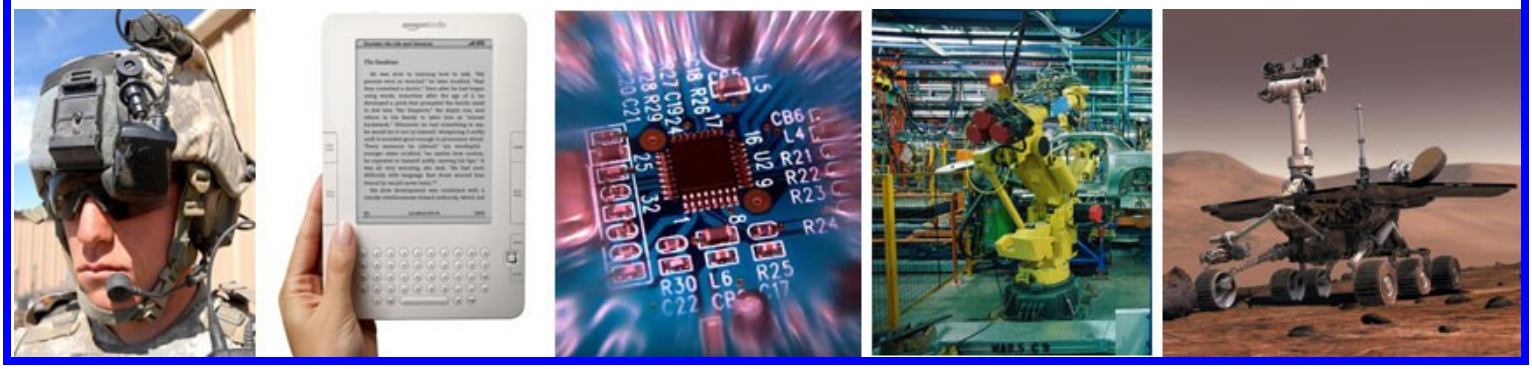
3rd Workshop on

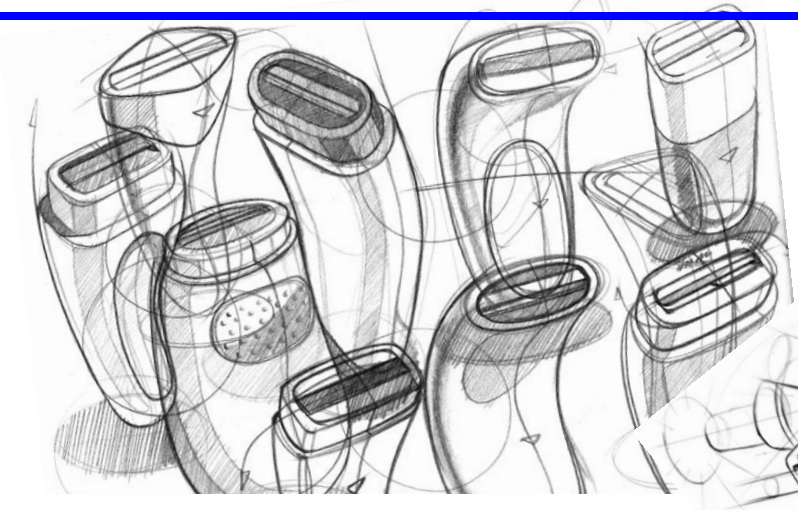
Analytic Virtual Integration for Cyber-Physical Systems

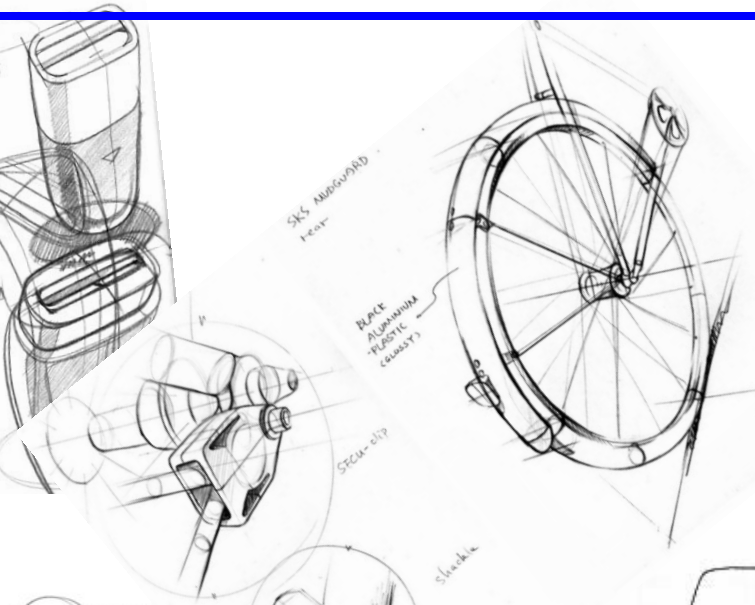
AVICPS 2012

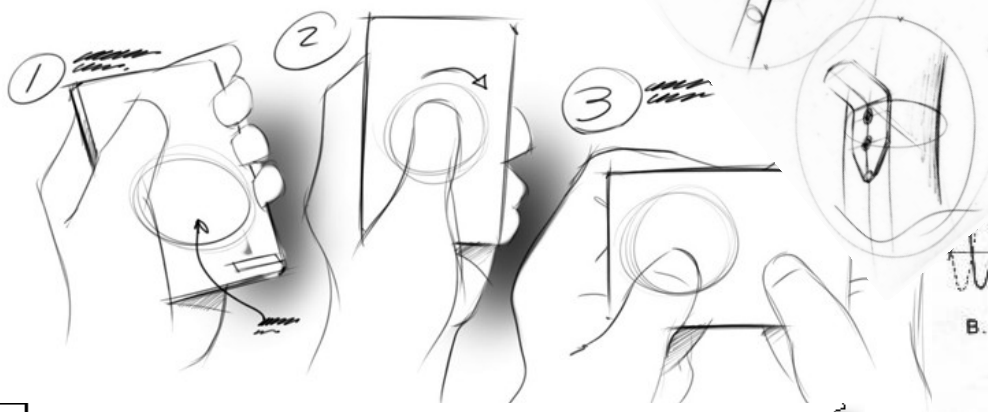
In conjunction with IEEE RTSS, San Juan, Puerto Rico

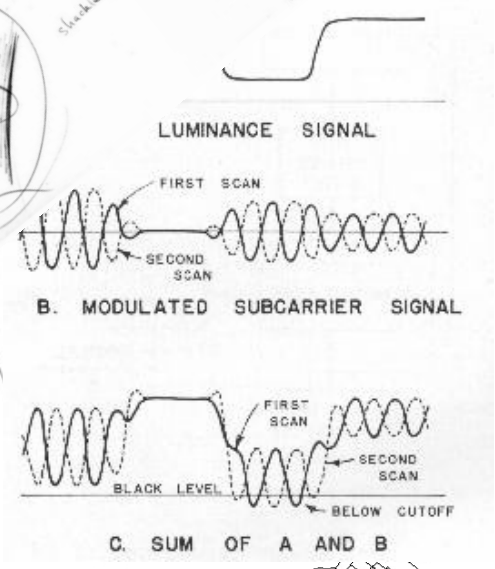
December 4th, 2012

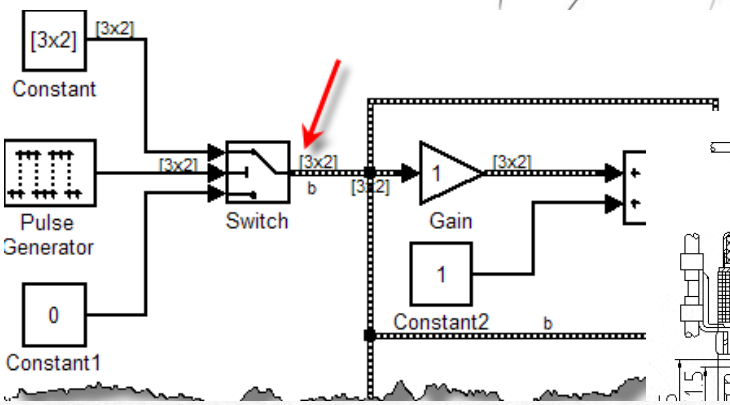


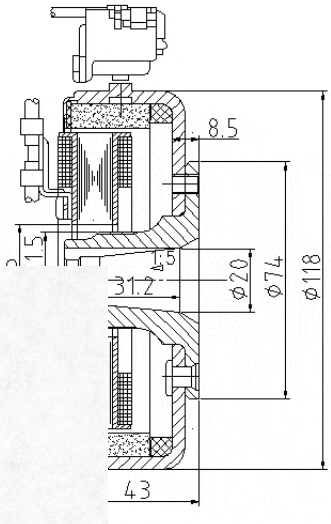


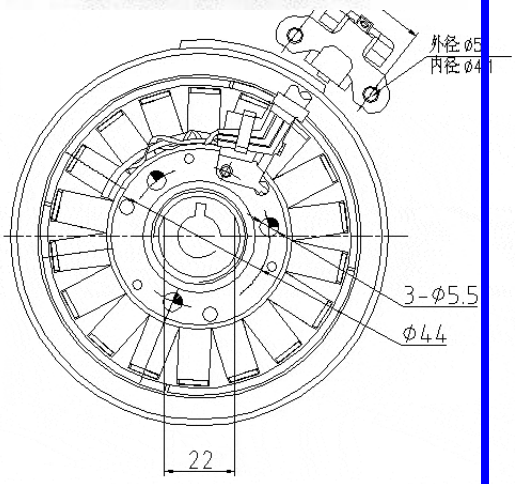


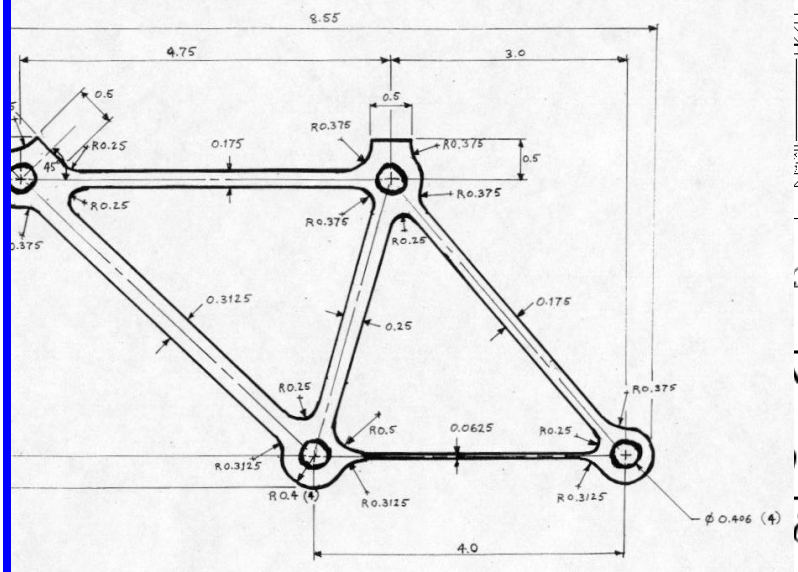












NOTES
1. ALL DIMENSIONS IN INCHES.
2. THICKNESS 0.25"
3. MATERIAL: AL 6061-T6

ACROBAT
SCALE: 1":1"

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

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

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

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