May 1, 2001

**Automotive Challenge Problems v.3.1**

**Introduction and Objectives**

This document is provided as a means to report to the MoBIES Automotive OEP community the current state of interactions between Phase I, Phase II, automotive industry and Government participants. It is intended to serve as a basis for the upcoming step-up in dialog between all MoBIES participants in order to meet the program goal of a coordinated Automotive OEP plan by the mid-July 01 PI meeting.

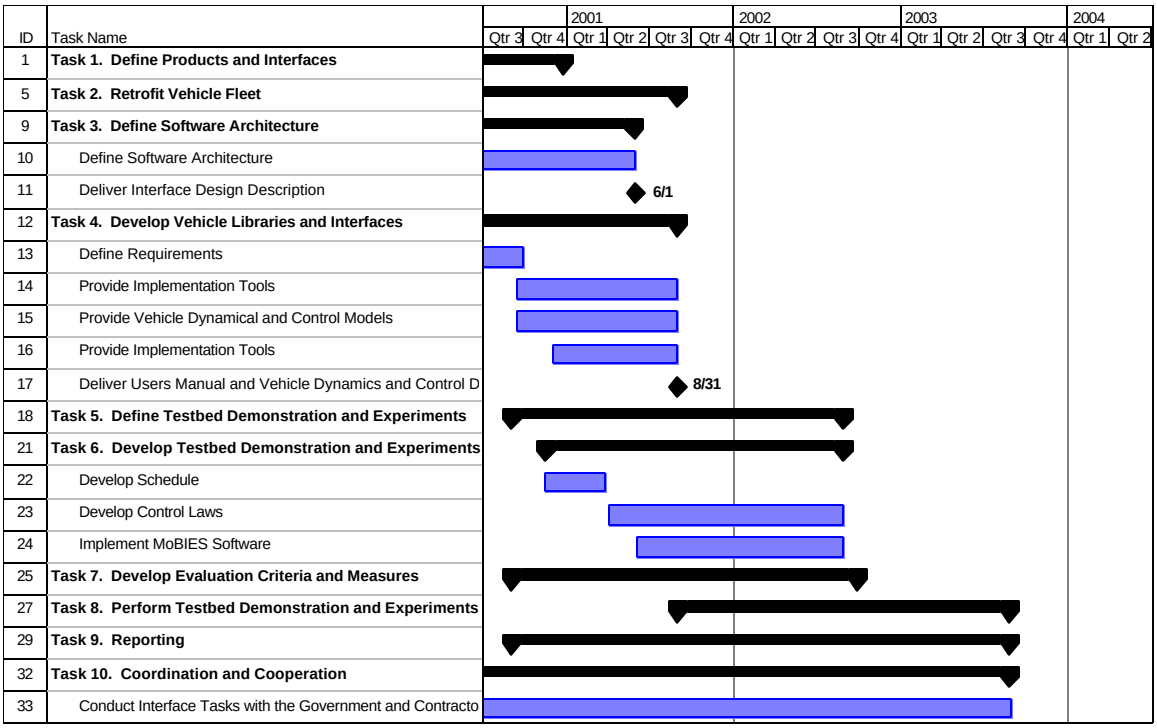
Hence, we request the following:

1. Careful consideration and feedback of the contents of this document. We ask you to particularly focus on *actions*, that is, how you would alter our interpretation of the challenge problems, if our expressed understanding of your contribution is accurate and finally, how you would structure within your MoBIES work scope a specific plan of action to participate in the Automotive OEP.
2. Communication with us. We have listed points of contact. Shortly, those points of contact will be in communication with you. However, if you wish please proact with us. Again, July 01 is rapidly approaching.
3. Development of your portion of an Automotive OEP program plan. Our intent in contacting you is to jointly develop the following:
   a. Your portion of the Automotive OEP Scope of Work. This means a narrative task-by-task description of how you will accomplish your objectives within the Phase II work.
   b. A schedule associated with 3a. Interim milestones will be part of that schedule (e.g., reporting certain prescribed events at PI meetings).
   c. Defined interfaces, types of data to be exchanged and deliverables also associated with 3a.
4. Iteration with us to ensure that coordination between all participants is accommodated. This means a give-and-take between Phase I, Phase II and other participants to ensure, within the scope of our contracts, that we can jointly produce the MoBIES program goals.

This document can be considered a starting point, but obviously to complete actions 1 – 4 by July will require us to move quickly from this document to interacting with each other, and finally to consensus on how to jointly proceed. Weaving together our approaches in a coordinated fashion will produce an interesting and worthwhile product, and we look forward to working with you.

**Berkeley Schedule**

As a reference, the UC Berkeley schedule is provided:




Note that Tasks 2 and 4 are UC Berkeley's responsibility (with considerable assistance from our automotive industry partners) and are covered by the present work, referenced in this document, whereas Tasks 5 and 6 are to be jointly developed.

What is most relevant is that Task 8, performing the mid-term experiment, commences in early Fall 01; that is our initial target, and we are developing schedule details in both the vehicle-vehicle communication and powertrain components of the OEP to achieve the start date. As we formalize our schedule during the earlier-referenced actions 1 – 4 we hope to provide you with sufficient detail that you can understand interfaces and products – essentially, we want to work with you to accommodate all aspects of the program.

Past the inception of Task 8, you will notice a relatively long period for that task. This means that several interim, detailed experiments can be performed along the way to highlight the particular technologies you purvey. Again, we look forward to mutually defining how they can be experimentally derived.

**Outline of Challenge Problems**

The remainder of this document revises the previous challenge problems from the automotive OEP. The revision incorporates discussions at the April 2001 ESWG meeting. The focus is on two applications: Powertrain Control (PC) and Cooperative Adaptive Cruise Control with Collision Warning (CACC+CW). The problems are divided into categories associated with a phase in a model-based design and implementation of embedded systems.

For each challenge problem there are

- a description;
- pointers to documents that provide detail;
- a contact person who can respond to questions;
- names of Phase I people who expressed interest in it; and
- our understanding of what they can contribute.

To meet the goals of the July 2001 PI meeting, Phase I people need to work on one or more of these problems. So far, only Edward Lee has responded to v.1 of this document, see Mobies Position Paper: http://vehicle.me.berkeley.edu/mobies/papers/Ptolemy.pdf

**Table of Challenge Problems**


1. Modeling
  1.1 Multiple-view modeling
  1.2 Automated composition of sub-components
  1.3 Communication models
2. Model Analysis
  2.1 Automatic test generation
  2.2 Verification
  2.3 Synthesis of switching
  2.4 Performance
3. Implementation
  3.1 Test vector generation
  3.2 Schedulability analysis
  3.3 Code generation
  3.4 Code debugging and testing
  3.5 RTOS generation
  3.6 Allocation to distributed platforms
4. Integration
  4.1 Model translation
  4.2 Integration of different models of computation
  4.3 Tool Integration
  4.4 Software/hardware Integration


## 1. Modeling

### 1.1 Multiple-view modeling


**Primary point of contact**: Ken Butts (mailto:kbutts1@ford.com), Mark Wilcutts (mailto:wilcutts@me.berkeley.edu)


The problem is to generate and relate plant and controller models at three levels [Butts]:

- level 1: hybrid automata with continuous dynamics
- level 2: discrete-time controllers and some scheduling information
- level 3: platform (e.g., OS, hardware) specific information (e.g., variable sizes).

Other refinements might include a more realistic communication model (see problem 1.3).

The questions are:

- how to "move" from one level to the next, e.g., perhaps automatically refine a level-1 model to a level-2 model
- how to preserve consistency (and what does that mean)

Links to detailed documents:

In http://vehicle.me.berkeley.edu/mobies/papers/challenges_berkeley.doc reference is made to a relevant paper by Magner, Butts, and Toeppe.

http://vehicle.me.berkeley.edu/mobies/papers/challenges_berkeley_v2.pdf

The modeling style guide at http://vehicle.me.berkeley.edu/mobies/papers/stylev242.pdf provides documentation of the Level 2 model structure currently used at Ford.

**Phase I response: Edward Lee**. Ptolemy II supports a hierarchical refinement of simulation models. At level 1, the plant can be represented by continuous odes, the controller by a sampled data system. At level 2, the level 1 controller can be embedded into an RTOS domain model to simulate the competition for system resources. For the CACC+CW problem, the model can be further extended to simulate network communication.

**Our understanding:** This means that to use Ptolemy II facilities the automotive plant and control models have to be rewritten in Ptolemy II. Moreover, to estimate the performance of the code on OSEK, one must simulate within Ptolemy the various control tasks and OSEK. These are very difficult tasks for the OEP group. Will the Ptolemy group undertake these tasks?

**Phase I response: John Anton(?)** said that the controller design normally conducted in the continuous-time world of odes, should also have attached to it the computational resources that the design would need. Then the control system performance (measured in the ode world) can be traded off against the computational resources of its implementation.

**Our understanding:** Perhaps this means that the controller design expressed at levels 1 and 2, should also simultaneously include level 3 considerations. The OEP control designs are formulated in Simulink/Stateflow and in Teja. Will Kestrel provide the means to attach (infer) the computational resource requirements that the designs imply?

**Additional note:** The study http://vehicle.me.berkeley.edu/mobies/papers/teja-simulink.pdf compares level 1, 2 implementations of the continuous-time system

$$\dot{x} = 1, \dot{y} = x; x(0) = y(0) = 0, 0 \le t \le 8.$$

This is the level 1 model. We are given the **explicit** level 2 constraint that $x$ must be computed every 1 second and $y$ every 1.5 seconds. The document compares the level 2 models (incorporating these constraints) in Teja, Simulink, and Simulink/Stateflow (following Ford style guide) in terms of

- how natural it is to go from level 1 to level 2;
- compactness of the level 2 models;
- approximation of the computation of resulting $x$, $y$ trajectories relative to the model 1 trajectories;

- efficiency of the implementation in terms of the compactness and readability of the resulting code.

Our conclusions are (1) that Teja scores better in all four dimensions. Furthermore, if one is only interested in simulation of the level 1 system, then Shift is the easiest to use.

### 1.2 Automated composition of sub-components

**Primary point of contact:** Bill Milam (mailto:wmilam@ford.com), Stavros Tripakis (mailto:stavros@EECS.Berkeley.EDU)

The problem is to find a method to build a specified target system by composing a given set of sub-components (e.g. block diagrams in Simulink).

Links to detailed documents:
http://vehicle.me.berkeley.edu/mobies/papers/model_composition_challenge.pdf
http://vehicle.me.berkeley.edu/mobies/papers/model_compiler.pdf both by B. Milam and A. Chutinan.
http://vehicle.me.berkeley.edu/mobies/papers/amc.pdf by S. Tripakis.

The first document motivates and describes the problem. There are two elements in a formal problem description. First, how is a component given (answer: it is a Simulink block). Second, what properties are relevant to a component's capability of being composed with other components (answer: signal type, suitably annotated, execution criteria (ODE solver parameters such as step size).

The second document proposes a "grammar" that (1) gives the rules for composing two components, and (2) infers the port description of the composed system from its components and the rule used.

The third document proposes an abstract formalization based on the second document. A component is described as a collection of input-output ports. Each port has a syntactical description (name, type, sampling rate, etc.). The "grammar" is expressed as a compositional relation ***C*** on ports. There are, in addition, fan-in and fan-out restrictions on port signals. There also are restrictions on the numbers of copies of a given component one is allowed to use. Given the target system **T** (specified in terms of *its* input-output ports), find an interconnection of components that realizes **T**, while satisfying ***C*** and the other restrictions, and minimizing some cost criterion. The formal problem is an integer programming problem. The complexity is exponential. However, under some conditions, the complexity is polynomial.

**Phase I response: Edward Lee**. The problem is specified in a declarative mode, i.e. two components may be connected if their input and output ports meet a generalized type constraint. The problem formulated in this way is likely to lead to too many solutions or no solution. A better approach is to have a hard-coded "model generator" that starts from the target system, and generates a pre-defined structure in terms of components. Those components may be parameterized (possibly in terms of the existing components?), and the designer fills in the appropriate parameters. Ptolemy II provides one example of the second approach: a high-order differential equation model (the target system) automatically generates a Simulink-style structure comprising first-order integration blocks.

**Our understanding:** Lee's "generative" approach is a special case of the generative grammar sketched in section 6 of the second document by Milam and Chutinan. One writes a target component **T** as (say) **T = (A** + **B)G**, where **A**, **B, G** are components and `+' and `.' denote particular types of port connection. If **A**, **B, G** are given components, we are done. Otherwise, we must realize them in terms of other components. Ultimately one obtains a realization of **T**. The difficulty with this approach, as Milam and Chutinan note, is that we don't know how to "expand" **T** so that we can effectively obtain a realization. The third document by Tripakis is at attempt to automate this expansion.

**Other Phase I response: none.**

### 1.3 Design and use of good (wireless) communication models

Primary point of contact: Pravin Varaiya (mailto:varaiya@EECS.Berkeley.EDU)

Automotive systems use inter-processor communication, e.g., micro-controllers communicating over a CAN bus. Telematics applications (e.g. CCAS+CW) require more complex networking infrastructure (both in terms of media, e.g., wireless, and protocols, e.g., TCP/IP). Communications are an important part of the design: they may restrict control performance; the latter may impose communications requirements. However, in the initial control design it is assumed that the modules communicate instantaneously and perfectly.

The goal is to develop simple communication models that are relevant for control design. Another goal is to develop control design methods that take into account communication system performance. These models can be used either for analysis or simulation.

Links to detailed documents:
See IEEE Control Systems Magazine, vol.21, (no.1), IEEE, Feb. 2001, for several articles dealing with networked-control system design.

**Phase I response: Edward Lee**. Ptolemy II is an excellent platform for modeling network communications.

**Our understanding:** One would have to develop a library of communications network simulation models, together with models of plant and controller design within Ptolemy. This daunting task cannot be undertaken by the OEP group. One alternative is to use existing simulation packages such as ns and Opnet. However, this poses the problem of integrating these packages with, say, Simulink or Teja that describe the plant and controller. (See challenge problem 4, below). Another approach is to build an adequate model within Simulink or Teja.

**Phase I response: I. Lee (U. Penn)** expressed interest in this problem.

## 2. Model Analysis

### 2.1 Automatic test generation

**Primary point of contact:** Stavros Tripakis (stavros@EECS.Berkeley.EDU)

The problem of automatic test generation is, given the model of a system (in some formalism, e.g., hybrid automata, Simulink), and a specification of the test goal, to generate a set of test cases that check whether the system meets the test goal.

The test cases are automata that act as observers/controllers to the system: they generate inputs to the system, and observe its outputs for some finite time. During this time interval they give a verdict, whether the system has passed or failed the test.

Automatic test generation can be viewed as "intelligent simulation." The objective is to generate a reasonable number of test cases that covers a representative class of behaviors, among all possible environment behaviors.

Links to detailed documents:
http://vehicle.me.berkeley.edu/mobies/papers/embedded_challenge.pdf

**Phase I response: Edward Lee**. Utility functions can be added to existing Ptolemy II to generate reports on test coverage at individual component and component interaction levels. Creation of testbenches, i.e. models that test other models, can also be supported.

**Our understanding:** Running simulation models of the design against typical plant behaviors tests Level 1 and level 2 control designs. In the PC design, one simulates typical loads, temperature, etc. to evaluate powertrain performance. In the CCAV+CW design, one simulates "typical" scenarios of inter-vehicle distance and speed, etc. The design team selects the test scenarios. Testing of code poses more difficult challenges that we haven't resolved.

**Phase I response: I. Lee (U. Penn)** expressed interest in "intelligent simulation."

### 2.2 Verification

**Primary point of contact: Pravin Varaiya** (varaiya@EECS.Berkeley.EDU)

The problem is to verify that a controller design in Simulink or Teja satisfies a given specification, for example, "an unsafe state is never reached", "the controller is never deadlocked", a variable used by the controller has been defined, and so on.

In the CACC+CW application, the main property to be verified is that collision between vehicles is avoided, that is, the distance between the subject vehicle and the vehicle in front is never zero.

In the PC application the unsafe or undesirable states might be specified by bounds on engine speed, fuel-air ratio, stability of idle speed, etc.

Links to detailed documents:

http://vehicle.me.berkeley.edu/mobies/papers/embedded_challenge.pdf

For a general introduction to the hierarchical control architecture see:
Varaiya, "Smart cars on smart roads," IEEE Trans Control, 38(2): 195-207,

Feb. 1993.

For a survey of control designs see:
Horowitz and Varaiya, "Control design of an automated highway system," Proc. IEEE, 88(7): 913-25, July 2000.

For verification see:
Puri and Varaiya, "Driving safely in smart cars," Proc. 1995 American Control Conference.

Botchkarev and Tripakis, "Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations", in Hybrid Systems: Computation and Control, LNCS 1790: 73-88, 2000. The tool discussed in this paper is available at http://robotics.eecs.berkeley.edu/~olegb/VeriSHIFT/

Kurzhanski and Varaiya, "Ellipsoidal techniques for reachability analysis", in Hybrid Systems: Computation and Control, LNCS 1790: 202-214, 2000.

**Phase I response: Edward Lee**. Tom Henzinger's group is working to integrate verifiable models, like Giotto, with Ptolemy II.

**Phase I respone: B. Krogh (CMU), I. Lee, R. Alur (U. Penn)** expressed interest in this problem.

**Our understanding:** Existing tools for verification of hybrid systems place strong restrictions on the system dynamics, which preclude their use for the automotive OEP. So one must resort to approximations. The use of FSM model-checking tools requires even further approximations. It would be valuable to see how the CMU and U. Penn tools work on the (non-hybrid) example in Puri and Varaiya.

**2.3 Synthesis of switching (hybrid) controllers**

**Primary point of contact:** Pravin Varaiya (mailto:varaiya@EECS.Berkeley.EDU)

The problem here is, given a set of macro-states (system modes), for each of which a control law is defined, and a set of switching conditions between these states, to synthesize a global controller which operates in any of these states and switches between them according to the conditions. The objectives are that the controller is stable, transitions are "smooth", and so on.

The synthesis might involve restricting the conditions, adding resets (re-initialize some variables), or synthesizing a transient set of states through which the controller passes during the switch.

Links to detailed documents:
For some work in this area, see:

Asarin et al, "Effective sysnthesis of switching controllers for linear systems," Proc. IEEE, 88(7): 1011-25, July 2000.

Koo et al, "Mode switching synthesis for reachability specifications," in Hybrid Systems: Computation and Control, LNCS, 2001.

**Phase I response: J. Koo (U. Penn).** The paper cited above provides an efficient computation of the mode-switching conditions.

**Our understanding:** Suppose you are given modes 1, …, N. Each mode $i$ is characterized by a given "safe" set $S_i$ and a differential equation $\dot{x} = f_i(x_i)$. You are told whether for every initial state in $S_i$ there is a trajectory (in mode $i$) that reaches $S_j$. The paper's procedure determines the sequence of mode transitions from an initial set $S_0$ to a final set $S_f$. This is obviously a graph search problem with N nodes. The difficulty in any actual application such as CCAC+CW is to determine if there indeed is a trajectory from $S_i$ to $S_j$.

**Phase I response: G. Pappas and R. Alur** expressed interest in this problem.

**2.4 Performance**

**Primary point of contact: Karl Hedrick** (mailto:khedrick@me.berkeley.edu)

The problem is to study robustness to parameter changes (sensitivity), fault tolerance, etc. Controller designs typically incorporate strategies for detection and reaction to faults.

Links to detailed documents:
For one study of how faults are included in controller design, see

Godbole et al, "Design and Verification of Communication Protocols for Degraded Modes of Operation of AHS," Proc. IEEE CDC, 427-32, 1995.

**Phase I response: B. Williams (MIT)** expressed interest in fault tolerance.

**Phase I response: Edward Lee**. The Ptolemy II simulation environment can be used to quickly prototype concepts of fault detection and isolation, and to integrate those models with those of the rest of the plant and controllers.

**Our understanding:** There are two steps in how control designs address fault tolerance. The first step involves *fault detection*. One assumes a set of models that describe the system under various fault conditions. The set includes the no-fault model. A separate controller is built for each fault condition. Based on sensor measurements, an on-line statistical procedure infers when a fault occurs and what type it is, and a "supervisor" switches in the controller built to handle that fault. There is a variety of inference procedures and redundant architectures to make robust the inference and fault-handling controllers.

**3. Implementation**

**3.1 Test vector generation**

**Primary point of contact: Ken Butts** (mailto:kbutts1l@ford.com), Stavros Tripakis (mailto:stavros@EECS.Berkeley.EDU)

This problem is related to problem 2.1, with the difference that it is not the function of the model which is exercised by the test vectors, but rather we are verifying the behavior of the implementation by comparing it to the behavior of the model (code and perhaps also hardware).

Links to detailed documents:
http://vehicle.me.berkeley.edu/mobies/papers/embedded_challenge.pdf

**Phase I response: I. Lee (U. Penn)** may expressed interest in this problem.

### 3.2 Schedulability analysis

**Primary point of contact: Stavros Tripakis** (mailto:stavros@EECS.Berkeley.EDU)

Most systems consist of a number of logical tasks. Each task is characterized by a set of activation conditions, execution time, resources that it has to access, and completion deadline. Upon implementation, these logical tasks are mapped onto one or more processes running on a single host machine, and sharing the CPU and other resources. Schedulability analysis consists in finding a scheduling policy to use for the processes so that the deadlines of the logical tasks are met (plus other properties such as absence of deadlocks, process starvation, and so on). Alternatively, given a scheduling policy, to determine whether these conditions are met.

We distinguish between logical and (physical) processes, since in general, more than one logical task can be implemented in the same process, where they are scheduled internally (e.g., Teja generates code like that). Even in this case, it is the requirements of the logical tasks that have to be met.

A particular challenge problem is to carry out in an automated way a schedulability analysis similar to the one described in the document below, for the publish/subscribe database architecture used in the automotive OEP. Part of the challenge is to come up with automated ways to estimate the various execution times necessary in the analysis. Even better would be a synthesis procedure that proposes how priorities are to be assigned to the different processes.

Links to detailed documents:
http://vehicle.me.berkeley.edu/mobies/vehicle/papers/pub-sub.pdf by Tripakis gives a preliminary analysis of the longitudinal and lateral control using the publish and subscribe architecture.
http://vehicle.me.berkeley.edu/mobies/vehicle/papers/taxys-cdc.pdf by Tripakis and Yovine gives another analysis of the same system.

The source code for the P&S architecture is available at:
http://vehicle.me.berkeley.edu/mobies/vehicle/PS-distribution.tar.gz

http://vehicle.me.berkeley.edu/mobies/papers/embedded_challenge.pdf is a more general document from Ford.

**Phase I response: Edward Lee.** A key problem in scheduling is that most methods are not compositional. Processes (and threads) consume shared resources in a complicated manner. So if process A and B can be accommodated separately, there is no easy way to ensure that A and B together can be accommodated. A TDM scheduler like Giotto and TTA simplifies schedulability since it divides CPU resources into time slots and assigns a time slot to each periodic task.

**Phase I response: B. Abbott (SWRI), R. Rajkumar (CMU), K. Shin (U. Mich)** expressed interest in this problem.

**Our understanding:** Traditional schedulability analysis like RMA is limited. Some limitations are overcome by extensions, eg., Harbour, Lehoczky, and Klein: "Analysis of tasks with varying fixed priorities," Prof. 12th IEEE Real-time Systems Symposioum, 1991. The above-cited document by Tripakis does this. Yet another approach based on Esterel and Kronos is presented in the document by

Tripakis and Yovine. Going to a TDM system certainly simplifies schedulability analysis. However, there may be a large cost: the underlying hardware and OS must support TDM; the fixed TDM schedule reduces flexibility; TDM schedules may not work for event-driven systems as in the PC problem where camshaft-driven events are very important.

### 3.3 Code generation

**Primary point of contact: Dave Bostic** (mailto:dbostic@ford.com), Paul Griffiths (mailto:pggriffi@vehicle.me.berkeley.edu)

The problem is to automatically generate code for a given platform, starting from a model (e.g., hybrid automata, dataflow blocks), so that the generated code preserves the properties of the model, under assumptions on the underlying platform. In the PC application, this is OSEK, MPC555+HC08. For the CACC+CW application, this is a publish and subscribe architecture on QNX.

Code generation can occur at various granularities: generating code for pieces of the entire model (e.g., Simulink blocks) up to generating code for the entire model (e.g., Teja). In the first case, support is necessary for "gluing" the pieces together (e.g., scheduling). In the latter case, support is necessary for schedulability analysis (c.f. problem 3.2). In case this analysis shows that some deadlines are missed, it is likely that this is due to the granularity of some atomic actions, which is too coarse (i.e., preemption of these actions is necessary). The tool should be able to figure this out and guide the user into splitting the actions in question into more fine-grain pieces.

Links to detailed documents:
http://vehicle.me.berkeley.edu/mobies/papers/embedded_challenge.pdf

**Phase I response: Edward Lee.** Ptolemy II can generate code at shallow and deep levels. Shallow code in Java uses Ptolemy libraries to execute a simulation. Deep code that targets specific designs (platforms like OSEK+MPC55?) can in principle be generated.

**Phase I response: John Anton (Kestrel)** expressed interest in this problem.

**Our understanding:** Executable simulation (shallow) code seems like the code generated by, say, Simulink or Shift. Teja generates code for the publish and subscribe architecture and a forthcoming Teja compiler for OSEK will generate code for the MPC555 platform.

### 3.4 Code debugging and testing

**Primary point of contact: Baris Dundar** (mailto:dundar@eecs.berkeley.edu)

Code debugging and testing refers to first, the ability to run and debug the code with or without hardware in the loop; second, the ability to map the results to the model from which the code has been generated. For example, if an error occurs during the execution of the code, say a variable X grows above an acceptable limit, one should be able to check whether the same behavior can be reproduced in the model. If this is so, then the model is incorrect. Otherwise, either some of the assumptions of the underlying platform were violated (e.g., not enough CPU), or the code generator is incorrect.

A useful method for code debugging and is the annotation of the code with "self-examining" parts, for example, assertions about the timing, values of variables, and so on. This is often done manually, and a challenge is to generate such annotations automatically and provide support for the interpretation of the results.

Links to detailed documents:
[Baris will provide reference]

**Phase I response: none.**

### 3.5 RTOS generation

**Primary point of contact: Bill Milam** (mailto:wmilam@ford.com), Baris Dundar (mailto:dundar@eecs.berkeley.edu), Mike Bauer mailto:Mike.Bauer@motorola.com

Ford's definition of the challenge problem is as follows: given a target software and hardware architecture, the worst-case execution time for the embedded system code, and additional timing constraints, generate a custom RTOS that enables the target code to meet all the timing requirements and is the most efficient in ROM, RAM, and CPU usage.

Automatic generation of OSEK OIL files for Matlab/Simulink generated code can also be considered under this topic, as the latest version of Matlab cannot generate OIL files for OSEK applications. OSEK Implementation Language (OIL) aims to create an OSEK-compliant RTOS scaled to a specific application. For all OSEK applications OIL must be used to statically configure the application at compile time. OIL is used to select the scheduling policy, define the objects (like tasks, alarms, events, resources, counters, ISRs...etc) in an application and their attributes.

Links to detailed documents:
http://vehicle.me.berkeley.edu/mobies/papers/embedded_challenge.pdf

**Phase I response: Edward Lee.** Addressing this problem within the Ptolemy framework is on the agenda.

**Phase I response: no one else.**

### 3.6 Allocation of system function and performance to distributed platforms

**Primary point of contact: Mike Bauer** (mailto:Mike.Bauer@motorola.com), Mark Wilcutts (mailto:wilcutts@me.berkeley.edu)

Implementation is relative to a given platform, which includes hardware components such as computers/micro-controllers, sensors, actuators, communication devices and links, and software such as operating systems, device drivers, libraries, or middleware (e.g., Corba, Jini, Publish/Subscribe). Often the choice of the underlying platform has been fixed by other factors, but it may be the case that a number or alternatives are possible.

One challenge problem is therefore to provide methods for choosing a platform, given a description of the particular application or class of applications that the platform has to support. The description might be the detailed model of the

application, or some general characteristics such as sampling frequencies, desired throughput, and so on.

Assuming the platform and application are fixed, and the platform is distributed, a challenge is to support the user in deciding how to partition the different functions or tasks of the application to the different computers, micro-controllers, etc. Such feedback may be input to the code-generation tools, which will generate code for the different parts, as well as for interfacing these parts (e.g., through a network). The PC platform has two processors, the CACC+CW also has multiple processors.

Links to detailed documents:
http://vehicle.me.berkeley.edu/mobies/papers/partitioned_control.pdf

**Phase 1 response: K. Shin (U. Mich) and R. Rajkumar (CMU)** expressed interest in this problem.

## 4. Integration

This problem has several aspects. In the automotive OEP, the problem concerns merging the PC and CACC+CW applications) at

- the modeling, simulation and analysis level; and
- the implementation level.

The controllers for PC and CACC+CW are complementary: CACC+CW produces a desired acceleration/deceleration output, while PC receives acceleration as input and produces torque as output.

In the first stages of the project, models and implementations of the two applications will be developed using different formalisms and tools, and on different platforms.

**Phase 1 response: E. Lee (UCB) and G. Karsai (VU)** have expressed interest in this set of problems.

The integration challenge is to develop methods and tools to perform one or more of the following functions:

### 4.1 Model translation

To/from TEJA

**Primary point of contact: Anouck Girard** (mailto:anouck@eecs.berkeley.edu), Marco Zandonatti (mailto:marcoz@teja.com)

### 4.2 integration of models of computation

This includes studying different underlying models of computation of each tool, and resolving whether the underlying assumptions are compatible, and what fixes are needed for meaningful model comparison/integration.

**Primary point of contact: Anouck Girard** (mailto:anouck@eecs.berkeley.edu)

**4.3 Tool integration** (e.g., Simulink and Teja) so two sets of interacting models can be run in parallel

The challenge here is to preserve real-time properties and semantical meaning during the execution of both tools in parallel.
This may include studying different verification and model checking tools, and studying whether the tools can work on the same models.

**Primary point of contact: Anouck Girard** (mailto:anouck@eecs.berkeley.edu)

**4.4 Software/Hardware Integration**

This includes studying aspects of several of the above challenge problems:

- what is the best way to communicate between software and hardware (P/S database),
- and how to map the software onto the hardware (schedulability analysis, allocation to distributed platforms, code generation, debugging and testing for particular hardware platforms etc…)

**Primary point of contact: Anouck Girard** (mailto:anouck@eecs.berkeley.edu)

Links to detailed documents:
http://vehicle.me.berkeley.edu/mobies/papers/MoBIES_v2v.pdf

Concerning the model development, the primary points of contact are:

Vehicle control:
- Plant: Adam Howell (mailto:ahowell@vehicle.me.berkeley.edu)
- Controllers: Mike Drew (mailto:mdrew@vehicle.me.berkeley.edu)
-

Powertrain control:
- Plant: Jason Souder (mailto:jsouder@me.berkeley.edu), Mark Wilcutts
- Controllers: Mark Wilcutts (mailto:wilcutts@me.berkeley.edu), Jason Souder (Ken Butts regarding controller representation in Simulink/Stateflow)

**Phase I response: Edward Lee.** The issue to be faced is that tools are based on one or more models of computation (MoC). So we need to understand the MoCs and the interaction between them. Since Ptolemy II supports multiple MoCs, it can be used as a glue between them. If we can “wrap” a specific tool to make it behave like a dataflow component, it may be much easier to integrate them.