Mobies Position Paper (DRAFT 2 Version)

Johan Eker
Jörn Janneck
Tak-Kuen John Koo
Edward A. Lee, PI
Jie Liu

PTOLEMY GROUP

DEPARTMENT OF EECS
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 97720

Abstract

This document is responding to the MoBIES "Automotive Challenge Problems". It is prepared by the Mobies Phase 1 Berkeley team, whose project is entitled "Process-Based Software Components for Networked Embedded Systems." The problems posed in the "Automotive Challenge Problems" paper are addresses one by one and our views on the problems are presented.

A. Motivation

As part of the Mobies phase I effort at Berkeley, we are developing a software framework called Ptolemy II. Ptolemy II is a component-oriented modeling and design framework written in Java. It is intended primarily to facilitate experimentation with design techniques and methodologies. We believe that a number of the challenge problems posed by the Mobies phase I team at Berkeley have been already addressed in the Ptolemy project, and for a number of others, we have ideas that we expect will lead to a solution.

A.1 The Ptolemy Project

The Ptolemy project studies heterogeneous modeling, simulation, and design of concurrent systems. The focus is on embedded systems [Lee], particularly those that mix technologies, including for example analog and digital electronics, hardware and software, and electronics and mechanical devices. The focus is also on systems that are complex in the sense that they mix widely different operations, such as signal processing, feedback control, sequential decision making, and user interfaces.

[Lee] Edward A. Lee, "What's Ahead for Embedded Software?," *IEEE Computer*, September 2000, pp. 18-26

A.2 Paper Structure and Notations

This paper is a response to the problems posed in the “Automotive Challenge Problems” paper. We address the problems one by one in the section below. The problems are included for convenience and presented in italics in the introduction of every section. Sections in *italics* are written by the Berkeley Phase I team, and sections in roman are written by the Phase II team.

B. Modeling

B.1 Multiple-view Modeling

≺Problem Statement➢

This problem consists of generating and/or maintaining a consistent set of models for the same system, but at different levels of abstraction. We may call these different "views" of the same model.

In [Butts] three levels of abstraction are defined:
- 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 removing the abstraction of "perfect" inter-module communication which is typical, and replacing it by a more realistic communication model.

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 when moving automatically, or check consistency of two models developed manually, where consistency means, e.g., some type-compatibility between inputs and outputs in terms of data size, sampling rate etc.

≺Response➢

Solutions to this problem have two different approaches within the Ptolemy project, and they are *hierarchical refinement* and *multiple-view models.*

B.1.1 Hierarchical Refinement

Ptolemy supports incremental refinement of simulation models through the use of different models of computation. The complexity of the model may be increased step by step by extending the model hierarchy. Typically, for a control system, the initial model specifies only the controller and the process. The process maybe modeled as ordinary differential equations (ODEs), and the controller maybe be described using discrete difference equations.

However, this model does not capture many issues related to an actual implementation. The usual assumptions is that the execution time is negligible and that there is no computation or communication jitter. Of course, this is not the case in the real-world. When the controller is running on a real computer and on top of a real-time operating system (RTOS), it will compete with other tasks for resources, e.g. the CPU and I/O. This will give rise to input-output delays and variations in the sampling period. Furthermore, the actuators and the sensors are usually not directly connected to the controller, but instead some network is used for transferring data. The network is a common resource possibly shared by many other control loops.These loops compete for network bandwidth. We would like to capture the above properties so that we can predict the real behavior of the embedded system, and evaluate scheduling mechanisms and communication protocols in terms of applications performance.

A more accurate model would include a model of the real-time operating system and the network. This is done in two steps in Ptolemy II. First, to consider the real-time issues, we embed the controller designed in the basic model (i.e. the composite actor that contains the finite state machine and the subcontrollers) into an RTOS domain model to capture the effects of the interaction between the different tasks running concurrently on the system. The RTOS-domain supports the simulation of concurrent tasks competing for system resources. The composite controller actor built in the basic model only specifies the computational part of the controller. To actually reflect the implementation, another task, which models the I/O part of the controller, is added. This I/O task may compete for resources with other I/O operations running on the system.

The model can now be extended further by including a model of the network communication. This is done by using a discrete event domain at the top level, and introducing a network actor, which models the behavior of a given network protocol. In this process of refining the design, components modeled in early phases can be reused.

In this process of refining a design, designers need to gradually add design considerations to the existing model and migrate the control system from algorithms to implementation. Different design perspectives usually imply heterogeneous component interaction styles. It is desirable that a design environment can support multiple component interaction styles and the components designed in earlier phases can be reused under new interaction styles, so that the verified properties can be preserved as much as possible. We argue that integrating different models of computation will help decompose design perspectives and achieve elegant and reusable models.

B.1.2 Multiple-view Models

In hierarchical refinement, the more detailed model subsumes all aspects and functionality of the refined one -- the properties of the refined model logically supervene on the properties of the more detailed one. As mentioned above, in case the two models are not formally derived from one another, the challenge is to prove this supervenience relation, i.e. to check whether the refinements are, in fact, consistent with the more abstract description.

An alternative interpretation of the above challenge is that the different views are not, in fact, refinements of each other, but that they represent complementing descriptions of a systems, sometimes called *facets* or *aspects*. Composing facets gives rise to the following questions:

- Are they consistent with each other, i.e. is there a system that satisfies all descriptions in all facets?
- How do facets interact? What are the implications of specifications in one facet in terms of another?

Complete answers to these questions are in general not computable, but even partial answers may be very useful, and by constraining the facet descriptions one may even be able to compute complete answers for interesting special cases.

There is a substantial body of research on these issues conducted in the context of *Rosetta* (www.sldl.org). Rosetta is a specification language whose central concept is that of a facet. Even though Rosetta includes facilities for describing structural aspects of a system (composition and refinement), its main focus is to facilitate multi-view modeling in the above sense. The language and its semantics framework provide a formal setting for studying facet composition and interaction, and for answering the questions above.

From a Ptolemy perspective, Rosetta's contributions are seen as essentially complementary to Ptolemy's, the latter being focused primarily on the structural aspects of systems descriptions. It would thus be most interesting to integrate these two approaches.

≺Phase II summary of this response≻

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.*

≺Phase II understanding≻

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?*

≺Our phase I plan≻

As part of our phase 1 effort we will deliver:

- A mechanism for hierarchically modeling hybrid controllers where continuous-time models can be discretized at multiple independent sample rates. This gets us from “level 1” to “level 2” as posed by the challenge problem. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- A mechanism for hierarchically combining multiple modeling techniques, where for example a component representing a model of a software realization of a controller realized in an RTOS can be embedded in a continuous-time model of the plant and controller working together. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- A framework for building code generators from discretized models that is hierarchical, in that levels of the hierachy can be autonomously synthesized, and each level is synthesized to respect the abstraction semantics used at that level of the hierarchy. **Schedule: started. This is a big task. Planned first releasable version by mid 2002.**
- Note that we do not believe that modeling arbitrary tasks running under an RTOS is the right approach. We do not believe that constructing applications as arbitrary tasks running under an RTOS is the right approach. Instead, models are constructed using a principled model of computation, such as Giotto, or the new RTOS domain we

are working on (see the RTOS generation challenge problem). Thus, we do not plan to undertake the proposed tasks.

B.2 Automated composition of sub-components

≺Problem Statement➢

The problem here is to come up with an efficient method for automatically composing a set of sub-components (e.g. block diagrams in Simulink) in order to build another component. "

≺Response➢

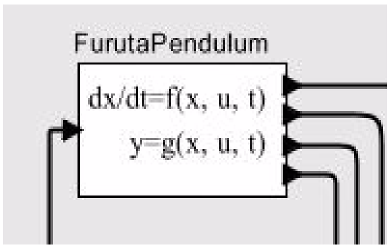
Automating component composition can be done in a variety of ways, which differ primarily in the type of specification that defines the composition. The three main approaches seem to be the following:

1. The specification is *declarative*, i.e. the modeler defines a set of constraints that define, e.g., compatibility relations between the ports of the components involved. The composition is automatically derived from these relations.

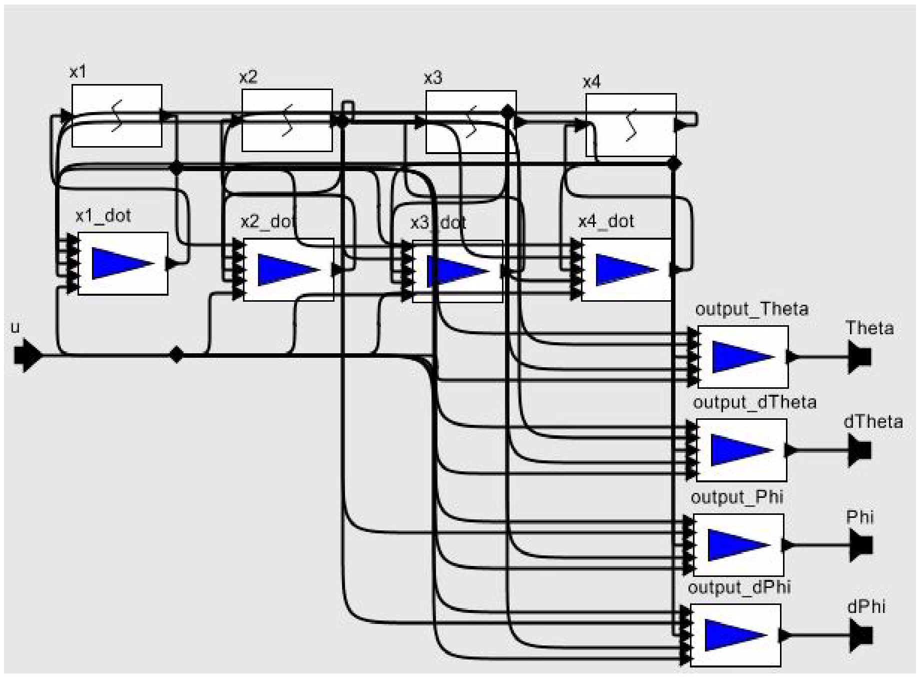
2. The modeler uses a set of hard-coded predefined *model generators*, that algorithmically create a model structure from some other input to them. This can be seen as a generalization of the first approach.

3. If the modeling language supports *parametric component construction* and/or *higher-order components*, the modeler may create model templates that essentially define partial model structures/compositions and can be parameterized to instantiate complete models.

The main problem with the first approach is that in general there may be any number of solutions to the constraint set, including zero and more than one. If there is no solution, it may not be immediately obvious precisely which constraint or combination of constraints cause the problem, so failure diagnosis and recovery may become an issue, particularly when the constraint set is large and highly interrelated. If there is more than one solution, choosing the right one, or alternatively ensuring that any is correct, may be far from trivial. One solution is to choose a declarative specification that has exactly one solution. However, it is unlikely that such a specification would be any more compact or understandable than a direct specification of the composition of sub-components, and hence would only amount to an alternative syntax for the same specification.



Ptolemy currently supports the second approach, where users may create components that generate and instantiate model structures depending on some parameters. For example, there is a model generator that can be parameterized with a set of differential equations. This component analyzes the equations and generates a corresponding block diagram that expresses the same relation between its inputs and its outputs as the equations. For example, the above component is expanded into the following model structure:



The third approach is an active research problem in the Ptolemy project. Where applicable, it is preferable to the second approach, because rather than having some algorithm create a potentially arbitrary model structure, it represents a more structured approach to automatic model creation. This in turn facilitates error detection, modular verification, and in general contributes significantly to the expressive power of the modeling language.

We intend to leverage existing approaches to higher-order visual modeling (cf. for instance the Moses project, www.tik.ee.ethz.ch/~moses), and adapt them to the requirements of the Ptolemy framework.

We believe that most issues arising from this challenge problem can be addressed by higher-order modeling components in conjunction with an expressive type system and a flexible visual syntax. In some special cases where these might not be adequate, we believe that it is important to have a general mechanism like the one currently implemented in Ptolemy.

≺Phase II summary of this response➢

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.*

≺Phase II understanding➢

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver:

- Components that synthesize complex models from algebraic descriptions of functionality. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- Components that synthesize complex models from particular combinators. **Schedule: started. We expect the first releasable version by mid 2002.**
- Tools that help in the construction of graphical models that follow regular patterns. These are visual renditions of the combinators above. **Schedule: planned. We expect the first releasable version by mid 2003.**

Note: We believe that name and/or type matching is most useful way to specify model structure when there is also a design pattern being applied. Our focus is on syntemizing the design pattern through the use of combinators.

Further note: We believe that generative approaches in general are about translating a specification in one language into a refinement in another. It is certainly possible to invent languages that we do not know how to translate. So let's avoid such languages.

B.3 Design and use of good (wireless) communication models

≺Problem Statement≻

Inter-module communication is already part of automotive systems, e.g., micro-controllers communicating over a CAN bus. With the introduction of applications requiring more complex networking infrastructure (both in terms of media, e.g., wireless, and in terms of protocols, e.g., TCP/IP), communications are an important part of the design. However, they are usually abstracted at the first level of the control design phase, where it is assumed that the modules communicate instantaneously and perfectly (no message loss).
The goal is to develop simple enough communication models, which are nevertheless relevant for control design. These models can be used either for analysis or simulation. Simple means not involving, for instance, a complete simulation of the protocol stack and channel models, as is typically done by a network simulator.

≺Response≻

The Ptolemy approach in this case is similar to the one presented in Hierarchical Refinement on page 3. Actors defined in previous simulations can be reused to model their behavior in a network setup. A typical example would be to model a distributed control system. In the first step, only the controller and the process are modeled as if they were directly connected to each other. This model is then extended by replacing the connections between the actors with actors that model the network.

Ptolemy provides an excellent platform for modeling of network communication for several reasons:

- The Ptolemy II type system supports composite types. In particular, a record type is a composition of named fields with values that are arbitrary types. Type constraints propogate transparently across operations that operate on these composite types. The record types can be used to aggregate data into packets that are then launched into abstracted communication subsystem models.
- Ptolemy also allows the user to define the type of the simulated messages as an ordinary Java class. The structure of the message could be represented in a high detail model containing headers, tails, CRC, etc., while in a a low resolution model only the data part is included.
- Real networks are designed in a hierarchical fashion with different layers having orthogonal and independent responsibilities. The lower layers handle the interaction with the physical world, i.e. transmitting and receiving packet, and manage data integrity, while the higher levels deal with session establishment, data routing and congestion resolution. The different characteristics of the layers make it suitable to model a network in a simulation framework that explicitly support different models of computation and their interaction. The interaction with the physical world requires continues time and event while session establishment is better expressed using finite state machines. The different levels could easily be refined and extended through different phases of the network modeling. While a simple network model only models random delays a more complex model takes packet collisions, error coding, bit errors, packet loss, and retransmissions into account.

≺Phase II summary of this response➢

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

≺Phase II understanding➢

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.*

➢Our phase I plan➢

As part of our phase 1 effort we will deliver:

- A framework that can simultaneously model communication networks (as in Opnet) and contollers and plants (as in Simulink), each using a modeling strategy suited to the problem being modeled. What we are delivering is ability to hierarchically compose distinct modeling strategies, not the libraries of modeling components that are needed to construct nontrivial network models. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- A framework that supports customization of the modeling semantics to match the realities of the communication network being used. For example, if a communication network with unreliable delivery is being used, then one might wish to construct a model of application by connecting components with unreliable communication links. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- Particular modeling semantics that tolerate communication latencies in communication systems by defining communication to be delayed. Giotto is one first example of such a modeling semantics. We are working on at least one other one that does not require the periodic structure of Giotto. **Schedule: started. Expect first versions released in mid 2002. Elaborations in 2003.**

C. Model Analysis

➢General Response➢

Challenge problems in this section advocate the use of formal models in designing, implementing, and testing embedded control systems, in particular, embedded software. Formal models and methods have long been used in control algorithm designs. For example, the formalism of linear/nonlinear systems, stability, controllability, observability, and robustness, are all based on solid mathematical foundations. However, traditionally, the design and implementation of embedded software is still full of ad hoc tricks and fragile twists. Theories, methodologies, and tools that help formalize embedded software models, analyze their properties, and simulate their real-world behaviors, are under high demand.

Many models exist in embedded software communities. However, some models are so coarse-grained, like publish and subscribe, that they should better be used as coordination models among processes and platforms. Some models are so fine-grained, like the original finite state machines, that using them to design complex systems becomes tedious and burdensome. Finding the right (patterns of) models of computation is a critical part of the Ptolemy project.

C.1 Automatic test generation

≺Problem Statement➢

The problem of automatic test generation is, given the model of a system (in some formalism, e.g., hybrid automata, Simulink blocks), 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 essentially automata that act as observers/controllers to the system: they generate inputs to the system, and observe the outputs of the system, for some finite time. At the end or before this time interval, they make 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 enough test cases (but also a reasonable number of them) that covers a representative enough class of behaviors, among all possible environment behaviors.

≺Response➢

We see testing at two levels -- the atomic level and the composite level. Almost all component-based design methodologies and tools, including Simulink, build complex systems using composition of atomic components. These atomic components may be provided by tool vendors or written by application designers. Testing of atomic components requests a certain amount of knowledge on how the component is written. Some tools place few constraints on how to write a component, which makes automated testing very difficult. On the other extreme, some tools restrict the model of building atomic component to obtain high testability. For example, in Polis [POLIS], components are written in a synchronous language, Esterel, and then compiled into codesign finite state machines (CFSM), which eases the testing and synthesis processes. However, restricted atomic component models sometimes bring less expressiveness and awkwardness on writing control algorithms. A study is undergoing on how to formalize models for Ptolemy II atomic actors. The models should both be intuitive to application designers, and expose enough formal properties for testing and analysis.

When atomic components are composed to form larger systems, the interaction styles among them become a critical part of the behavior of the system. Having formal models for component interaction also helps testing and analyzing the system. For example, if a piece of embedded software is built using Boolean Dataflow [BDF], then it may sometimes be possible to generate a sequence of inputs to test that all components in the system have been executed.

In terms of software infrastructure support, utility functions can be easily added to the existing Ptolemy II framework to generate reports on the test coverage at both atomic component level and component interaction level. The creation of testbenches, i.e. models that test other models, can also be easily supported.

[POLIS] F. Balarin, et. al., *Hardware-Software Co-Design of Embedded Systems, the POLIS Approach*, Kluwer Academic Publisher, 1997.

[BDF] Joe T. Buck and Edward A. Lee, *The Token Flow Model*, in Advanced Topics in Dataflow Computing and Multi-threading, ed. Lubomir Bic, Guang, Gao, and Jean-Luc Gaudiot, IEEE Computer Society Press, 1993

⮜Phase II summary of this response⮞

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.*

⮜Phase II understanding⮞

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 "typi-*

cal" 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.

⋖Our phase I plan⋗

We now interpret this challenge problem more broadly to be concerned with assurance. Debugging and testing methods are part of the solutions rather than part of the problem. Consider for example a component that needs new input on all ports in order to react. A proper design ensures that new input is available on all ports before a reaction is stimulated. There are three approaches to ensuring that a design is “proper”:

- a testing approach,
- an assertions approach, or
- a static analysis approach.

A testing approach checks everything at run time by testing for particular violations. An assertions approach statically declares, as part of the design, the parameters of correct behavior, and verifies at run time that these parameters are met. A static analysis approach is most familiar today in the form of a type system. Components declare their parameters of correctness as part of their interface definition, and a design time tool, such as a compiler, checks that these parameters are met. Formal verification techniques fall in this category as well, but our approach is much more like type systems than like model checking or theorem proving.

As part of our phase 1 effort we will deliver:

- A mechanism for defining dynamic properties of interfaces (such as that new inputs are required on all ports to react). **Schedule: first version done using FSMs and reported in http://ptolemy.eecs.berkeley.edu/publications/papers/00/systemLevel/. A second version interface automata (by Luca de Alfaro) has also been done, but not yet reported.**
- A mechanism for composing interface definitions to perform “type check” statically. **Schedule: started. First version expected by mid 2002. A complete theory and software support is expected by end 2003, perhaps.**

C.2 Verification

≺Problem Statement➢

The problem is to verify that a given model in a formalism such as the above 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.

≺Response➢

There are two related issues here -- correct by construction and verification. Many system design methodologies advocate correct by construction, that is, certain properties hold as long as the design staying within a framework. Pole placement in linear systems is an example of such a framework that ensures stability. In embedded software design, there are similar methodologies. For example, in synchronous dataflow models, deadlock-free is a property that can be statically analyzed. If we have a correct-by-construction framework, then verification is not an issue.

Certain properties for embedded systems may be hard to provide by correct-by-construction frameworks. In these cases, we also would like that the embedded software be built in formal models such that formal verification techniques, like reachability analysis and model checking, can be applied. Tom Henzinger’s group, part of our Mobies effort at UC Berkeley, is working to integrate verifiable models, like Giotto, with system design frameworks, like Ptolemy II.

≺Phase II summary of this response➢

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

≺Phase II understanding➢

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver:

- Generators that produce code that is “correct by construction” in that it matches the (narrow) semantics of a well-understood model that does not therefore require elaborate verification. Giotto and synchronous dataflow, for example, are modeling frameworks with sufficiently narrow semantics that strong properties can be asserted about any correct-by-construction implementation. **Schedule: started. First versions of code generators from Giotto and SDF are expected by mid 2002. Inclusion of FSM and RTOS is expected in 2003. Note that this is the same code generator promised above in B.1.2, Multiple-View Models.**

Note: One view of the restriction on system dynamics is that it “precludes use for the automotive OEP” and one must resort to approximation. Another view is that requires abstraction. We prefer the second view.

C.3 Synthesis of switching (hybrid) controllers

≺Problem Statement➢

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.

≺Response➢

This work has been performed in the SEC project at Berkeley.

In many control applications, a specific set of controllers of satisfactory performance have already been designed and must be used. When such a collection of control modes is available, an important problem is to be able to accomplish a variety of high level tasks by appropriately switching between the low-level control modes. In [KPS], a framework for determining the sequence of control modes satisfying reachability tasks is proposed. The approach consists of extracting a finite graph which refines the original collections of control modes, but is consistent with the physical system, in the sense that high level design has feasible implementation. Therefore, the control mode graph can then be used online for efficient and dependable real-time mode switching. For determining the switching conditions between different modes, there exists synthesis algorithm [ABDMP] for linear controlled systems. As shown in [KPS], if the closed loop dynamics are considered, the switching conditions between control modes can be computed efficiently by just examining the stability properties in the control modes. The framework presented in [KPS] is illustrated on a nonlinear helicopter control system with four control modes.

[KPS] T. J. Koo, G. J. Pappas, and S. Sastry, “Mode Switching Synthesis for Reachability Specifications,” Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, Springer, 2001.

[ABDMP] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli, “Effective Synthesis of Switching Controllers for Linear Systems,” In Proceedings of the IEEE, 88, Special Issue on Hybrid System: Theory & Applications, 1011-1025, 2000.

C.4 Performance

≺Problem Statement≻

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

≺Response≻

We acknowledge the studies on networked control systems, real-time performance of embedded controllers, and fault detection and isolation (FDI). We are not expecting to contribute on the theoretical aspects of these studies. However, we believe that modeling and simulation environments that allows people who work on such theories to quickly prototype their concepts is equally important. As stated in section B.1, Ptolemy II is able to integrate models for real-time scheduling, network protocols, controllers, FDI algorithms, and plant dynamics, and allow designers to gradually increment design considerations.

≺Phase II summary of this response➢

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.*

≺Phase II understanding➢

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver one or more techniques for constructing modal models. We will not specifically address the FDI problem, but we believe that this modal modeling infrastructure can be used to explore the problem.

- A finite-state machine domain that can be composed hierarchically with other domains to achieve modal models. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- An improved visual syntax for constructing modal models hierarchically. **Schedule: started. First version expected in mid 2002.**
- A generalized masking scheme where models that are assemblies of components have associated with them a state machine, where each state of the state machine “masks” the components that are active. **Schedule: We have constructed a first prototype as part of the SEC project, and may be pursuing this under that project.**

D. Implementation

D.1 Test vector generation

≺Problem Statement≻

They can be simulated and analyzed using a given tool: one challenge here is to compare the semantics and expressiveness of the different formalisms, and indicate which is more suitable for which typical control applications.

≺Not responded yet≻

D.2 Schedulability analysis

≺Problem Statement≻

Most systems consist of a number of logical tasks, where 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, therefore sharing the CPU and other resources. The problem of schedulability analysis consists in determining which policy to use for scheduling the physical tasks 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. Notice that we distinguish between logical and physical tasks (processes), since in general, more than one logical tasks 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 at PATH. Part of the challenge problem 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.

≺Response≻

Processes, and their cousin, threads, are widely used for concurrent software design. Indeed, processes can be viewed as a component technology, where a multitasking operating system or multithreaded execution engine provides the framework that coordinates the components. Component interaction mechanisms, monitors, semaphores, and remote procedure calls, are supported by the framework. In this context, a process can be viewed as a component that exposes at its interface an ordered sequence of external interactions. However, as a component technology, processes and threads are extremely weak. A composition of two processes is not a process (it no longer exposes at its interface an ordered sequence of external interactions). Worse, a composition of two processes is not a component of any sort that we can easily characterize. It is for this reason that concurrent programs built from processes or threads are so hard to get right. It is very difficult to talk about the properties of the aggregate because we have no ontology for the aggregate.

A key problem in scheduling is that most methods are not compositional. That is, even if assurances can be provided individually to a pair of components, there are no systematic mechanisms for providing assurances to two, except in trivial cases. A chronic problem with priority-based scheduling, known as priority inversion, is one manifestation of this problem.

≺Phase II summary of this response➢

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 II understanding➢

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.*

≺Our phase I plan≻

As part of our phase 1 effort we will deliver:

- Modeling semantics such as Giotto where schedulability analysis is greatly simplified by disciplining the inter-component interactions (data dependencies and synchronization). This will include modeling semantics that are applicable to other sorts of problems than Giotto, such as event-based problems. **Schedule: This is pretty open ended work, hard to pin down to a schedule. Under the SEC project, we have developed an RTOS domain in Ptolemy II that has semantics somewhere between Giotto and a conventional RTOS. We are studying this to see how to adapt it to the automotive OEP scenario.**

D.3 Code Generation

≺Problem Statement≻

The problem here is to automatically generate code for a given platform, starting from a model (e.g., hybrid automata, dataflow blocks), such that the generated code preserves the properties of the model, potentially under a number of assumptions on the underlying platform.

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, supports necessary for performing 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.

≺Response≻

The feasibility and quality of code generation heavily depends on the models that are used to create high-level designs. For example, to generate code that implements a discrete-event model of computation, an event queue and sorting algorithms have to be generated; to generate code that implements a discrete-time models, a time-triggered execution engine need to be generated; to generate prioritized preemptive models, prioritized process queues and preemption mechanism need to be generated, and so on. Mixing and matching right models of computation not only make complex designs more understandable, but also helps optimize generated code.

In Ptolemy II, code can be generated at two levels -- shallow and deep. Code generated by shallow code generation will rely on Ptolemy II packages to execute. The generated code still use Ptolemy actor libraries, actor composition, and directors. Although the implementation language is still Java, the code generation process can create an independent Java application and package only necessary Ptolemy II classes with it.

Deep code generation looks into individual actors, and creates abstract syntax trees (AST) for them. Together with knowledge provided by the type system and models of computation, it can generate (in principle) highly efficient codes that targets specific designs. Since the AST is implementation independent, the backend language can easily be C/C++. This work is still highly preliminary, but we are optimistic that it will yield a far better approach to code generation than any of the commonly used alternatives.

We are also looking at platform dependent run-time support for Ptolemy designs. See further discussion in section D.5.

≺Phase II summary of this response➢

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 II understanding➢

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver:

- A framework for building code generators from several models of computation, and example code generators at least for Giotto, SDF, and FSM. **Schedule: first versions in mid 2002. More complete versions in 2003.**

D.4 Code debugging and testing

≺Problem Statement➢

Code debugging and testing refers to: first, being able to run and debug the code with or without hardware in the loop; second, being able to map the results to the model from which the code has been generated (if this is the case). 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 has been 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.

≺Not responded yet.➢

D.5 RTOS generation

≺Problem Statement➢

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.

We think that automatic generation of OSEK OIL files for Matlab/Simulink generated code can also be considered under this topic, as the latest version of Matlab can not generate OIL files for OSEK applications. OSEK Implementation Language (OIL) aims to create an OSEK compli-

ant 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.

≺Response➢

We share the vision with Ford that embedded systems sometimes need application specific run-time environment, e.g. RTOS, to achieve high execution efficiency and low foot-print. We have studied run-time models for control-centric embedded systems and proposed a hierarchical composition of run-time models based on Ptolemy component architecture and models of computation [LJL]. Existing RTOSs usually only provide a flat layer of abstraction and one model of computation -- prioritized preemptive scheduling. All applications have to map to such a run-time model, regardless whether the model matches the application. One purpose of our hierarchical run-time models is to allow designers to use execution models that fit the application best and to deploy only the necessary run-time support packages.

In particular, we propose to solve the problem from two ends. From the design side, embedded system designers should be able to explore different models of computations to match the application. This will typically ends up with hierarchically composing heterogeneous models of computation. And from the implementation side, a run-time environment can be developed that support hierarchical composition of run-time models. Such a run-time environment will be platform dependent and highly (re)configurable. The run-time environment will also reduce the code generation complexity, since the implementation architectures match the modeling and design architectures.

[LJL] Jie Liu, Stan Jefferson, and Edward A. Lee, “Motivating Hierarchical Run-Time Models for Measurement and Control Systems,” to appear in 2001 American Control Conference (ACC’01), Arlington, VA, June 2001.

≺Phase II summary of this response≻

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

≺Our phase I plan≻

As part of our phase 1 effort we will deliver:

- Code generation for Giotto and possibly our RTOS domain. **Schedule: In 2003.**

D.6 System Partition

≺Problem Statement≻

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).

≺Not responded yet.➢

E. Integration

This is one challenge problem that has multiple aspects. It has to do with merging of different control applications (e.g the PC and CACC+CW applications):

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

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

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

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

E.1 Model translation

≺Problem Statement➢

They can be simulated and analyzed using a given tool: one challenge here is to compare the semantics and expressiveness of the different formalisms, and indicate which is more suitable for which typical control applications.

E.2 Integration of models and 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.

≺Response➢

We do not quite understand what exactly the problem statement is. The Ptolemy project is all about integration of models of computation.

E.3 Tool Integration

≺Problem Statement➢

Integrate tools (e.g., Simulink and Teja) so that they can simulate in conjunction two sets of interacting models. The challenge here is to preserve real-time properties during the execution of the two tools in parallel.

≺Response➢

One fundamental issue about (interactive) tool integration is the semantics integration. Tools are designed based on one or more models of computation. For tool integration, we have to understand the MoC and study the interaction among them. In most cases, this may require the tools to expose more information than it usually does.

For example, if we want to do mixed-signal simulation by integrating discrete-event simulators with continuous-time simulators, then we require that the discrete event simulator to expose its “next event time” in addition to its current time. Obviously, not all discrete event simulators support that. In an early study [LWLL], we claimed that tools are likely to be able to communicate with other tools with the same semantics. Based on the fact that Ptolemy II support multiple MoC, we can use Ptolemy II as a glue to integrate tools.

It is also helpful to look at tool integration from dataflow and control flow perspectives. At an abstract level, execution in a design tool can be view as data flow objects. They read inputs, do a finite computation, and produce outputs. Individual tools also apply specific control flows to its internal execution. They typically control how the messages are delivered from one component to another, what is the execution order among components, and what is an atomic piece of execution. The concepts of composite actor and hierarchy in Ptolemy II provide a polymorphic way of encapsulating control flow. Composite actors wrap their internal components execution in such a way that they behaves like atomic actors. Similarly, if we can wrap tools in such a way that they behave like (domain-polymorphic) dataflow components, then the integration of tools could be lot easier. Of course, this is not always the case. Sometimes, control flow information must be exposed to the outside.

[LWLL] Jie Liu, Bicheng Wu, Xiaojun Liu, and Edward A. Lee, “Inter-operation of Heterogeneous CAD Tools in Ptolemy II,” in *symposium on Design, Test, and Microfabrication of MEMS/MOEMs*, March 1999, Paris, France

E.4 Software/hardware integration

≺Problem Statement➢

One challenge here is to build interfaces through which the two implementations will communicate, or alternatively, re-generate the implementations, given that they will run in parallel.

≺Not responded yet.➢

PTOLEMY GROUP

DEPARTMENT OF EECS
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 97720

Mobies Position Paper (DRAFT 2 Version)

Johan Eker
Jörn Janneck
Tak-Kuen John Koo
Edward A. Lee, PI
Jie Liu

Abstract

This document is responding to the MoBIES "Automotive Challenge Problems". It is prepared by the Mobies Phase 1 Berkeley team, whose project is entitled "Process-Based Software Components for Networked Embedded Systems." The problems posed in the "Automotive Challenge Problems" paper are addresses one by one and our views on the problems are presented.

A. Motivation

As part of the Mobies phase I effort at Berkeley, we are developing a software framework called Ptolemy II. Ptolemy II is a component-oriented modeling and design framework written in Java. It is intended primarily to facilitate experimentation with design techniques and methodologies. We believe that a number of the challenge problems posed by the Mobies phase I team at Berkeley have been already addressed in the Ptolemy project, and for a number of others, we have ideas that we expect will lead to a solution.

A.1 The Ptolemy Project

The Ptolemy project studies heterogeneous modeling, simulation, and design of concurrent systems. The focus is on embedded systems [Lee], particularly those that mix technologies, including for example analog and digital electronics, hardware and software, and electronics and mechanical devices. The focus is also on systems that are complex in the sense that they mix widely different operations, such as signal processing, feedback control, sequential decision making, and user interfaces.

[Lee] Edward A. Lee, "What's Ahead for Embedded Software?," *IEEE Computer*, September 2000, pp. 18-26

A.2 Paper Structure and Notations

This paper is a response to the problems posed in the “Automotive Challenge Problems” paper. We address the problems one by one in the section below. The problems are included for convenience and presented in italics in the introduction of every section. Sections in *italics* are written by the Berkeley Phase I team, and sections in roman are written by the Phase II team.

B. Modeling

B.1 Multiple-view Modeling

≺Problem Statement➢

This problem consists of generating and/or maintaining a consistent set of models for the same system, but at different levels of abstraction. We may call these different "views" of the same model.

In [Butts] three levels of abstraction are defined:
- 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 removing the abstraction of "perfect" inter-module communication which is typical, and replacing it by a more realistic communication model.

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 when moving automatically, or check consistency of two models developed manually, where consistency means, e.g., some type-compatibility between inputs and outputs in terms of data size, sampling rate etc.

≺Response➢

Solutions to this problem have two different approaches within the Ptolemy project, and they are *hierarchical refinement* and *multiple-view models.*

B.1.1 Hierarchical Refinement

Ptolemy supports incremental refinement of simulation models through the use of different models of computation. The complexity of the model may be increased step by step by extending the model hierarchy. Typically, for a control system, the initial model specifies only the controller and the process. The process maybe modeled as ordinary differential equations (ODEs), and the controller maybe be described using discrete difference equations.

However, this model does not capture many issues related to an actual implementation. The usual assumptions is that the execution time is negligible and that there is no computation or communication jitter. Of course, this is not the case in the real-world. When the controller is running on a real computer and on top of a real-time operating system (RTOS), it will compete with other tasks for resources, e.g. the CPU and I/O. This will give rise to input-output delays and variations in the sampling period. Furthermore, the actuators and the sensors are usually not directly connected to the controller, but instead some network is used for transferring data. The network is a common resource possibly shared by many other control loops.These loops compete for network bandwidth. We would like to capture the above properties so that we can predict the real behavior of the embedded system, and evaluate scheduling mechanisms and communication protocols in terms of applications performance.

A more accurate model would include a model of the real-time operating system and the network. This is done in two steps in Ptolemy II. First, to consider the real-time issues, we embed the controller designed in the basic model (i.e. the composite actor that contains the finite state machine and the subcontrollers) into an RTOS domain model to capture the effects of the interaction between the different tasks running concurrently on the system. The RTOS-domain supports the simulation of concurrent tasks competing for system resources. The composite controller actor built in the basic model only specifies the computational part of the controller. To actually reflect the implementation, another task, which models the I/O part of the controller, is added. This I/O task may compete for resources with other I/O operations running on the system.

The model can now be extended further by including a model of the network communication. This is done by using a discrete event domain at the top level, and introducing a network actor, which models the behavior of a given network protocol. In this process of refining the design, components modeled in early phases can be reused.

In this process of refining a design, designers need to gradually add design considerations to the existing model and migrate the control system from algorithms to implementation. Different design perspectives usually imply heterogeneous component interaction styles. It is desirable that a design environment can support multiple component interaction styles and the components designed in earlier phases can be reused under new interaction styles, so that the verified properties can be preserved as much as possible. We argue that integrating different models of computation will help decompose design perspectives and achieve elegant and reusable models.

B.1.2 Multiple-view Models

In hierarchical refinement, the more detailed model subsumes all aspects and functionality of the refined one -- the properties of the refined model logically supervene on the properties of the more detailed one. As mentioned above, in case the two models are not formally derived from one another, the challenge is to prove this supervenience relation, i.e. to check whether the refinements are, in fact, consistent with the more abstract description.

An alternative interpretation of the above challenge is that the different views are not, in fact, refinements of each other, but that they represent complementing descriptions of a systems, sometimes called *facets* or *aspects.* Composing facets gives rise to the following questions:

- Are they consistent with each other, i.e. is there a system that satisfies all descriptions in all facets?
- How do facets interact? What are the implications of specifications in one facet in terms of another?

Complete answers to these questions are in general not computable, but even partial answers may be very useful, and by constraining the facet descriptions one may even be able to compute complete answers for interesting special cases.

There is a substantial body of research on these issues conducted in the context of *Rosetta* (www.sldl.org). Rosetta is a specification language whose central concept is that of a facet. Even though Rosetta includes facilities for describing structural aspects of a system (composition and refinement), its main focus is to facilitate multi-view modeling in the above sense. The language and its semantics framework provide a formal setting for studying facet composition and interaction, and for answering the questions above.

From a Ptolemy perspective, Rosetta's contributions are seen as essentially complementary to Ptolemy's, the latter being focused primarily on the structural aspects of systems descriptions. It would thus be most interesting to integrate these two approaches.

≺Phase II summary of this response≻

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.*

≺Phase II understanding≻

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?*

≺Our phase I plan≻

As part of our phase 1 effort we will deliver:

- A mechanism for hierarchically modeling hybrid controllers where continuous-time models can be discretized at multiple independent sample rates. This gets us from “level 1” to “level 2” as posed by the challenge problem. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- A mechanism for hierarchically combining multiple modeling techniques, where for example a component representing a model of a software realization of a controller realized in an RTOS can be embedded in a continuous-time model of the plant and controller working together. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- A framework for building code generators from discretized models that is hierarchical, in that levels of the hierachy can be autonomously synthesized, and each level is synthesized to respect the abstraction semantics used at that level of the hierarchy. **Schedule: started. This is a big task. Planned first releasable version by mid 2002.**
- Note that we do not believe that modeling arbitrary tasks running under an RTOS is the right approach. We do not believe that constructing applications as arbitrary tasks running under an RTOS is the right approach. Instead, models are constructed using a principled model of computation, such as Giotto, or the new RTOS domain we

are working on (see the RTOS generation challenge problem). Thus, we do not plan to undertake the proposed tasks.

B.2 Automated composition of sub-components

≺Problem Statement➢

The problem here is to come up with an efficient method for automatically composing a set of sub-components (e.g. block diagrams in Simulink) in order to build another component. “

≺Response➢

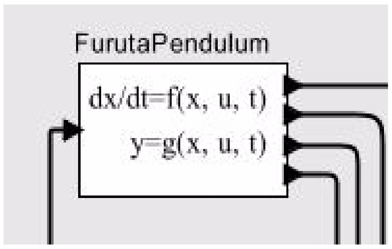
Automating component composition can be done in a variety of ways, which differ primarily in the type of specification that defines the composition. The three main approaches seem to be the following:

1. The specification is *declarative*, i.e. the modeler defines a set of constraints that define, e.g., compatibility relations between the ports of the components involved. The composition is automatically derived from these relations.

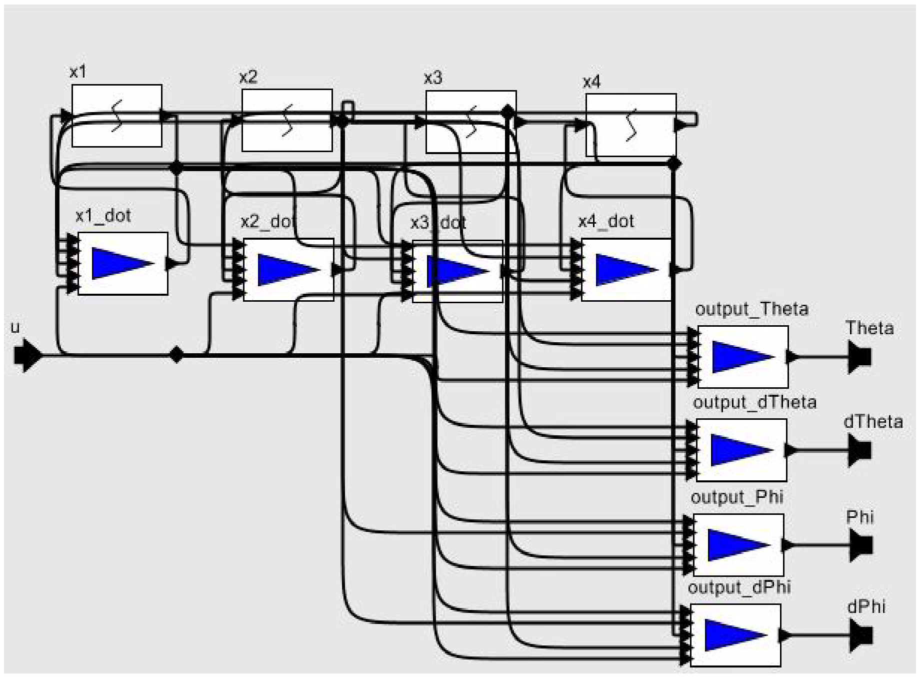
2. The modeler uses a set of hard-coded predefined *model generators*, that algorithmically create a model structure from some other input to them. This can be seen as a generalization of the first approach.

3. If the modeling language supports *parametric component construction* and/or *higher-order components*, the modeler may create model templates that essentially define partial model structures/compositions and can be parameterized to instantiate complete models.

The main problem with the first approach is that in general there may be any number of solutions to the constraint set, including zero and more than one. If there is no solution, it may not be immediately obvious precisely which constraint or combination of constraints cause the problem, so failure diagnosis and recovery may become an issue, particularly when the constraint set is large and highly interrelated. If there is more than one solution, choosing the right one, or alternatively ensuring that any is correct, may be far from trivial. One solution is to choose a declarative specification that has exactly one solution. However, it is unlikely that such a specification would be any more compact or understandable than a direct specification of the composition of sub-components, and hence would only amount to an alternative syntax for the same specification.



Ptolemy currently supports the second approach, where users may create components that generate and instantiate model structures depending on some parameters. For example, there is a model generator that can be parameterized with a set of differential equations. This component analyzes the equations and generates a corresponding block diagram that expresses the same relation between its inputs and its outputs as the equations. For example, the above component is expanded into the following model structure:



The third approach is an active research problem in the Ptolemy project. Where applicable, it is preferable to the second approach, because rather than having some algorithm create a potentially arbitrary model structure, it represents a more structured approach to automatic model creation. This in turn facilitates error detection, modular verification, and in general contributes significantly to the expressive power of the modeling language.

We intend to leverage existing approaches to higher-order visual modeling (cf. for instance the Moses project, www.tik.ee.ethz.ch/~moses), and adapt them to the requirements of the Ptolemy framework.

We believe that most issues arising from this challenge problem can be addressed by higher-order modeling components in conjunction with an expressive type system and a flexible visual syntax. In some special cases where these might not be adequate, we believe that it is important to have a general mechanism like the one currently implemented in Ptolemy.

≺Phase II summary of this response➢

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.*

≺Phase II understanding➢

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver:

- Components that synthesize complex models from algebraic descriptions of functionality. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- Components that synthesize complex models from particular combinators. **Schedule: started. We expect the first releasable version by mid 2002.**
- Tools that help in the construction of graphical models that follow regular patterns. These are visual renditions of the combinators above. **Schedule: planned. We expect the first releasable version by mid 2003.**

Note: We believe that name and/or type matching is most useful way to specify model structure when there is also a design pattern being applied. Our focus is on syntemizing the design pattern through the use of combinators.

Further note: We believe that generative approaches in general are about translating a specification in one language into a refinement in another. It is certainly possible to invent languages that we do not know how to translate. So let’s avoid such languages.

B.3 Design and use of good (wireless) communication models

≺Problem Statement≻

Inter-module communication is already part of automotive systems, e.g., micro-controllers communicating over a CAN bus. With the introduction of applications requiring more complex networking infrastructure (both in terms of media, e.g., wireless, and in terms of protocols, e.g., TCP/IP), communications are an important part of the design. However, they are usually abstracted at the first level of the control design phase, where it is assumed that the modules communicate instantaneously and perfectly (no message loss).
The goal is to develop simple enough communication models, which are nevertheless relevant for control design. These models can be used either for analysis or simulation. Simple means not involving, for instance, a complete simulation of the protocol stack and channel models, as is typically done by a network simulator.

≺Response≻

The Ptolemy approach in this case is similar to the one presented in Hierarchical Refinement on page 3. Actors defined in previous simulations can be reused to model their behavior in a network setup. A typical example would be to model a distributed control system. In the first step, only the controller and the process are modeled as if they were directly connected to each other. This model is then extended by replacing the connections between the actors with actors that model the network.

Ptolemy provides an excellent platform for modeling of network communication for several reasons:

- The Ptolemy II type system supports composite types. In particular, a record type is a composition of named fields with values that are arbitrary types. Type constraints propogate transparently across operations that operate on these composite types. The record types can be used to aggregate data into packets that are then launched into abstracted communication subsystem models.
- Ptolemy also allows the user to define the type of the simulated messages as an ordinary Java class. The structure of the message could be represented in a high detail model containing headers, tails, CRC, etc., while in a a low resolution model only the data part is included.
- Real networks are designed in a hierarchical fashion with different layers having orthogonal and independent responsibilities. The lower layers handle the interaction with the physical world, i.e. transmitting and receiving packet, and manage data integrity, while the higher levels deal with session establishment, data routing and congestion resolution. The different characteristics of the layers make it suitable to model a network in a simulation framework that explicitly support different models of computation and their interaction. The interaction with the physical world requires continues time and event while session establishment is better expressed using finite state machines. The different levels could easily be refined and extended through different phases of the network modeling. While a simple network model only models random delays a more complex model takes packet collisions, error coding, bit errors, packet loss, and retransmissions into account.

≺Phase II summary of this response➢

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

≺Phase II understanding➢

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver:

- A framework that can simultaneously model communication networks (as in Opnet) and contollers and plants (as in Simulink), each using a modeling strategy suited to the problem being modeled. What we are delivering is ability to hierarchically compose distinct modeling strategies, not the libraries of modeling components that are needed to construct nontrivial network models. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- A framework that supports customization of the modeling semantics to match the realities of the communication network being used. For example, if a communication network with unreliable delivery is being used, then one might wish to construct a model of application by connecting components with unreliable communication links. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- Particular modeling semantics that tolerate communication latencies in communication systems by defining communication to be delayed. Giotto is one first example of such a modeling semantics. We are working on at least one other one that does not require the periodic structure of Giotto. **Schedule: started. Expect first versions released in mid 2002. Elaborations in 2003.**

C. Model Analysis

≺General Response➢

Challenge problems in this section advocate the use of formal models in designing, implementing, and testing embedded control systems, in particular, embedded software. Formal models and methods have long been used in control algorithm designs. For example, the formalism of linear/nonlinear systems, stability, controllability, observability, and robustness, are all based on solid mathematical foundations. However, traditionally, the design and implementation of embedded software is still full of ad hoc tricks and fragile twists. Theories, methodologies, and tools that help formalize embedded software models, analyze their properties, and simulate their real-world behaviors, are under high demand.

Many models exist in embedded software communities. However, some models are so coarse-grained, like publish and subscribe, that they should better be used as coordination models among processes and platforms. Some models are so fine-grained, like the original finite state machines, that using them to design complex systems becomes tedious and burdensome. Finding the right (patterns of) models of computation is a critical part of the Ptolemy project.

C.1 Automatic test generation

≺Problem Statement➢

The problem of automatic test generation is, given the model of a system (in some formalism, e.g., hybrid automata, Simulink blocks), 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 essentially automata that act as observers/controllers to the system: they generate inputs to the system, and observe the outputs of the system, for some finite time. At the end or before this time interval, they make 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 enough test cases (but also a reasonable number of them) that covers a representative enough class of behaviors, among all possible environment behaviors.

≺Response➢

We see testing at two levels -- the atomic level and the composite level. Almost all component-based design methodologies and tools, including Simulink, build complex systems using composition of atomic components. These atomic components may be provided by tool vendors or written by application designers. Testing of atomic components requests a certain amount of knowledge on how the component is written. Some tools place few constraints on how to write a component, which makes automated testing very difficult. On the other extreme, some tools restrict the model of building atomic component to obtain high testability. For example, in Polis [POLIS], components are written in a synchronous language, Esterel, and then compiled into codesign finite state machines (CFSM), which eases the testing and synthesis processes. However, restricted atomic component models sometimes bring less expressiveness and awkwardness on writing control algorithms. A study is undergoing on how to formalize models for Ptolemy II atomic actors. The models should both be intuitive to application designers, and expose enough formal properties for testing and analysis.

When atomic components are composed to form larger systems, the interaction styles among them become a critical part of the behavior of the system. Having formal models for component interaction also helps testing and analyzing the system. For example, if a piece of embedded software is built using Boolean Dataflow [BDF], then it may sometimes be possible to generate a sequence of inputs to test that all components in the system have been executed.

In terms of software infrastructure support, utility functions can be easily added to the existing Ptolemy II framework to generate reports on the test coverage at both atomic component level and component interaction level. The creation of testbenches, i.e. models that test other models, can also be easily supported.

[POLIS] F. Balarin, et. al., *Hardware-Software Co-Design of Embedded Systems, the POLIS Approach*, Kluwer Academic Publisher, 1997.

[BDF] Joe T. Buck and Edward A. Lee, *The Token Flow Model*, in Advanced Topics in Dataflow Computing and Multi-threading, ed. Lubomir Bic, Guang, Gao, and Jean-Luc Gaudiot, IEEE Computer Society Press, 1993

⋞Phase II summary of this response⋟

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.*

⋞Phase II understanding⋟

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 "typi-*

cal" 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.

≺Our phase I plan➢

We now interpret this challenge problem more broadly to be concerned with assurance. Debugging and testing methods are part of the solutions rather than part of the problem. Consider for example a component that needs new input on all ports in order to react. A proper design ensures that new input is available on all ports before a reaction is stimulated. There are three approaches to ensuring that a design is “proper”:

- a testing approach,
- an assertions approach, or
- a static analysis approach.

A testing approach checks everything at run time by testing for particular violations. An assertions approach statically declares, as part of the design, the parameters of correct behavior, and verifies at run time that these parameters are met. A static analysis approach is most familiar today in the form of a type system. Components declare their parameters of correctness as part of their interface definition, and a design time tool, such as a compiler, checks that these parameters are met. Formal verification techniques fall in this category as well, but our approach is much more like type systems than like model checking or theorem proving.

As part of our phase 1 effort we will deliver:

- A mechanism for defining dynamic properties of interfaces (such as that new inputs are required on all ports to react). **Schedule: first version done using FSMs and reported in http://ptolemy.eecs.berkeley.edu/publications/papers/00/systemLevel/. A second version interface automata (by Luca de Alfaro) has also been done, but not yet reported.**
- A mechanism for composing interface definitions to perform “type check” statically. **Schedule: started. First version expected by mid 2002. A complete theory and software support is expected by end 2003, perhaps.**

C.2 Verification

≺Problem Statement≻

The problem is to verify that a given model in a formalism such as the above 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.

≺Response≻

There are two related issues here -- correct by construction and verification. Many system design methodologies advocate correct by construction, that is, certain properties hold as long as the design staying within a framework. Pole placement in linear systems is an example of such a framework that ensures stability. In embedded software design, there are similar methodologies. For example, in synchronous dataflow models, deadlock-free is a property that can be statically analyzed. If we have a correct-by-construction framework, then verification is not an issue.

Certain properties for embedded systems may be hard to provide by correct-by-construction frameworks. In these cases, we also would like that the embedded software be built in formal models such that formal verification techniques, like reachability analysis and model checking, can be applied. Tom Henzinger’s group, part of our Mobies effort at UC Berkeley, is working to integrate verifiable models, like Giotto, with system design frameworks, like Ptolemy II.

≺Phase II summary of this response≻

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

≺Phase II understanding≻

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver:

- Generators that produce code that is "correct by construction" in that it matches the (narrow) semantics of a well-understood model that does not therefore require elaborate verification. Giotto and synchronous dataflow, for example, are modeling frameworks with sufficiently narrow semantics that strong properties can be asserted about any correct-by-construction implementation. **Schedule: started. First versions of code generators from Giotto and SDF are expected by mid 2002. Inclusion of FSM and RTOS is expected in 2003. Note that this is the same code generator promised above in B.1.2, Multiple-View Models.**

Note: One view of the restriction on system dynamics is that it "precludes use for the automotive OEP" and one must resort to approximation. Another view is that requires abstraction. We prefer the second view.

C.3 Synthesis of switching (hybrid) controllers

≺Problem Statement➢

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.

≺Response➢

This work has been performed in the SEC project at Berkeley.

In many control applications, a specific set of controllers of satisfactory performance have already been designed and must be used. When such a collection of control modes is available, an important problem is to be able to accomplish a variety of high level tasks by appropriately switching between the low-level control modes. In [KPS], a framework for determining the sequence of control modes satisfying reachability tasks is proposed. The approach consists of extracting a finite graph which refines the original collections of control modes, but is consistent with the physical system, in the sense that high level design has feasible implementation. Therefore, the control mode graph can then be used online for efficient and dependable real-time mode switching. For determining the switching conditions between different modes, there exists synthesis algorithm [ABDMP] for linear controlled systems. As shown in [KPS], if the closed loop dynamics are considered, the switching conditions between control modes can be computed efficiently by just examining the stability properties in the control modes. The framework presented in [KPS] is illustrated on a nonlinear helicopter control system with four control modes.

[KPS] T. J. Koo, G. J. Pappas, and S. Sastry, “Mode Switching Synthesis for Reachability Specifications,” Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, Springer, 2001.

[ABDMP] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli, “Effective Synthesis of Switching Controllers for Linear Systems,” In Proceedings of the IEEE, 88, Special Issue on Hybrid System: Theory & Applications, 1011-1025, 2000.

C.4 Performance

≺Problem Statement➢

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

≺Response➢

We acknowledge the studies on networked control systems, real-time performance of embedded controllers, and fault detection and isolation (FDI). We are not expecting to contribute on the theoretical aspects of these studies. However, we believe that modeling and simulation environments that allows people who work on such theories to quickly prototype their concepts is equally important. As stated in section B.1, Ptolemy II is able to integrate models for real-time scheduling, network protocols, controllers, FDI algorithms, and plant dynamics, and allow designers to gradually increment design considerations.

≺Phase II summary of this response➢

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.*

≺Phase II understanding➢

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver one or more techniques for constructing modal models. We will not specifically address the FDI problem, but we believe that this modal modeling infrastructure can be used to explore the problem.

- A finite-state machine domain that can be composed hierarchically with other domains to achieve modal models. **Schedule: done. This is part of Ptolemy II v. 1.0.**
- An improved visual syntax for constructing modal models hierarchically. **Schedule: started. First version expected in mid 2002.**
- A generalized masking scheme where models that are assemblies of components have associated with them a state machine, where each state of the state machine “masks” the components that are active. **Schedule: We have constructed a first prototype as part of the SEC project, and may be pursuing this under that project.**

D. Implementation

D.1 Test vector generation

≺Problem Statement≻

They can be simulated and analyzed using a given tool: one challenge here is to compare the semantics and expressiveness of the different formalisms, and indicate which is more suitable for which typical control applications.

≺Not responded yet≻

D.2 Schedulability analysis

≺Problem Statement≻

Most systems consist of a number of logical tasks, where 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, therefore sharing the CPU and other resources. The problem of schedulability analysis consists in determining which policy to use for scheduling the physical tasks 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. Notice that we distinguish between logical and physical tasks (processes), since in general, more than one logical tasks 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 at PATH. Part of the challenge problem 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.

≺Response≻

Processes, and their cousin, threads, are widely used for concurrent software design. Indeed, processes can be viewed as a component technology, where a multitasking operating system or multithreaded execution engine provides the framework that coordinates the components. Component interaction mechanisms, monitors, semaphores, and remote procedure calls, are supported by the framework. In this context, a process can be viewed as a component that exposes at its interface an ordered sequence of external interactions. However, as a component technology, processes and threads are extremely weak. A composition of two processes is not a process (it no longer exposes at its interface an ordered sequence of external interactions). Worse, a composition of two processes is not a component of any sort that we can easily characterize. It is for this reason that concurrent programs built from processes or threads are so hard to get right. It is very difficult to talk about the properties of the aggregate because we have no ontology for the aggregate.

A key problem in scheduling is that most methods are not compositional. That is, even if assurances can be provided individually to a pair of components, there are no systematic mechanisms for providing assurances to two, except in trivial cases. A chronic problem with priority-based scheduling, known as priority inversion, is one manifestation of this problem.

≺Phase II summary of this response➢

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 II understanding➢

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.*

≺Our phase I plan≻

As part of our phase 1 effort we will deliver:

- Modeling semantics such as Giotto where schedulability analysis is greatly simplified by disciplining the inter-component interactions (data dependencies and synchronization). This will include modeling semantics that are applicable to other sorts of problems than Giotto, such as event-based problems. **Schedule: This is pretty open ended work, hard to pin down to a schedule. Under the SEC project, we have developed an RTOS domain in Ptolemy II that has semantics somewhere between Giotto and a conventional RTOS. We are studying this to see how to adapt it to the automotive OEP scenario.**

D.3 Code Generation

≺Problem Statement≻

The problem here is to automatically generate code for a given platform, starting from a model (e.g., hybrid automata, dataflow blocks), such that the generated code preserves the properties of the model, potentially under a number of assumptions on the underlying platform.

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, supports necessary for performing 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.

≺Response≻

The feasibility and quality of code generation heavily depends on the models that are used to create high-level designs. For example, to generate code that implements a discrete-event model of computation, an event queue and sorting algorithms have to be generated; to generate code that implements a discrete-time models, a time-triggered execution engine need to be generated; to generate prioritized preemptive models, prioritized process queues and preemption mechanism need to be generated, and so on. Mixing and matching right models of computation not only make complex designs more understandable, but also helps optimize generated code.

In Ptolemy II, code can be generated at two levels -- shallow and deep. Code generated by shallow code generation will rely on Ptolemy II packages to execute. The generated code still use Ptolemy actor libraries, actor composition, and directors. Although the implementation language is still Java, the code generation process can create an independent Java application and package only necessary Ptolemy II classes with it.

Deep code generation looks into individual actors, and creates abstract syntax trees (AST) for them. Together with knowledge provided by the type system and models of computation, it can generate (in principle) highly efficient codes that targets specific designs. Since the AST is implementation independent, the backend language can easily be C/C++. This work is still highly preliminary, but we are optimistic that it will yield a far better approach to code generation than any of the commonly used alternatives.

We are also looking at platform dependent run-time support for Ptolemy designs. See further discussion in section D.5.

≺Phase II summary of this response➢

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 II understanding➢

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.*

≺Our phase I plan➢

As part of our phase 1 effort we will deliver:

- A framework for building code generators from several models of computation, and example code generators at least for Giotto, SDF, and FSM. **Schedule: first versions in mid 2002. More complete versions in 2003.**

D.4 Code debugging and testing

≺Problem Statement≻

Code debugging and testing refers to: first, being able to run and debug the code with or without hardware in the loop; second, being able to map the results to the model from which the code has been generated (if this is the case). 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 has been 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.

≺Not responded yet.≻

D.5 RTOS generation

≺Problem Statement≻

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.

We think that automatic generation of OSEK OIL files for Matlab/Simulink generated code can also be considered under this topic, as the latest version of Matlab can not generate OIL files for OSEK applications. OSEK Implementation Language (OIL) aims to create an OSEK compli-

ant 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.

≺Response➢

We share the vision with Ford that embedded systems sometimes need application specific run-time environment, e.g. RTOS, to achieve high execution efficiency and low foot-print. We have studied run-time models for control-centric embedded systems and proposed a hierarchical composition of run-time models based on Ptolemy component architecture and models of computation [LJL]. Existing RTOSs usually only provide a flat layer of abstraction and one model of computation -- prioritized preemptive scheduling. All applications have to map to such a run-time model, regardless whether the model matches the application. One purpose of our hierarchical run-time models is to allow designers to use execution models that fit the application best and to deploy only the necessary run-time support packages.

In particular, we propose to solve the problem from two ends. From the design side, embedded system designers should be able to explore different models of computations to match the application. This will typically ends up with hierarchically composing heterogeneous models of computation. And from the implementation side, a run-time environment can be developed that support hierarchical composition of run-time models. Such a run-time environment will be platform dependent and highly (re)configurable. The run-time environment will also reduce the code generation complexity, since the implementation architectures match the modeling and design architectures.

[LJL] Jie Liu, Stan Jefferson, and Edward A. Lee, “Motivating Hierarchical Run-Time Models for Measurement and Control Systems,” to appear in 2001 American Control Conference (ACC’01), Arlington, VA, June 2001.

≺Phase II summary of this response≻

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

≺Our phase I plan≻

As part of our phase 1 effort we will deliver:

- Code generation for Giotto and possibly our RTOS domain. **Schedule: In 2003.**

D.6 System Partition

≺Problem Statement≻

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).

≺Not responded yet.➢

E. Integration

This is one challenge problem that has multiple aspects. It has to do with merging of different control applications (e.g the PC and CACC+CW applications):

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

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

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

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

E.1 Model translation

≺Problem Statement➢

They can be simulated and analyzed using a given tool: one challenge here is to compare the semantics and expressiveness of the different formalisms, and indicate which is more suitable for which typical control applications.

E.2 Integration of models and 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.

≺Response➢

We do not quite understand what exactly the problem statement is. The Ptolemy project is all about integration of models of computation.

E.3 Tool Integration

≺Problem Statement≻

Integrate tools (e.g., Simulink and Teja) so that they can simulate in conjunction two sets of interacting models. The challenge here is to preserve real-time properties during the execution of the two tools in parallel.

≺Response≻

One fundamental issue about (interactive) tool integration is the semantics integration. Tools are designed based on one or more models of computation. For tool integration, we have to understand the MoC and study the interaction among them. In most cases, this may require the tools to expose more information than it usually does.

For example, if we want to do mixed-signal simulation by integrating discrete-event simulators with continuous-time simulators, then we require that the discrete event simulator to expose its “next event time” in addition to its current time. Obviously, not all discrete event simulators support that. In an early study [LWLL], we claimed that tools are likely to be able to communicate with other tools with the same semantics. Based on the fact that Ptolemy II support multiple MoC, we can use Ptolemy II as a glue to integrate tools.

It is also helpful to look at tool integration from dataflow and control flow perspectives. At an abstract level, execution in a design tool can be view as data flow objects. They read inputs, do a finite computation, and produce outputs. Individual tools also apply specific control flows to its internal execution. They typically control how the messages are delivered from one component to another, what is the execution order among components, and what is an atomic piece of execution. The concepts of composite actor and hierarchy in Ptolemy II provide a polymorphic way of encapsulating control flow. Composite actors wrap their internal components execution in such a way that they behaves like atomic actors. Similarly, if we can wrap tools in such a way that they behave like (domain-polymorphic) dataflow components, then the integration of tools could be lot easier. Of course, this is not always the case. Sometimes, control flow information must be exposed to the outside.

[LWLL] Jie Liu, Bicheng Wu, Xiaojun Liu, and Edward A. Lee, "Interoperation of Heterogeneous CAD Tools in Ptolemy II," in *symposium on Design, Test, and Microfabrication of MEMS/MOEMs*, March 1999, Paris, France

E.4 Software/hardware integration

≺Problem Statement➢

One challenge here is to build interfaces through which the two implementations will communicate, or alternatively, re-generate the implementations, given that they will run in parallel.

≺Not responded yet.➢