Submitted for publication.

Multi-domain Modeling of Cyber-Physical Systems Using Architectural Views

Ajinkya Bhave, Bruce Krogh
Dept. of Electrical & Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15217
{jinx—krogh}@ece.cmu.edu

David Garlan, Bradley Schmerl
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15217
{garlan—schmerl}@cs.cmu.edu

***Abstract*—Designing cyber-physical systems (CPSs) increasingly requires the use of multi-domain models throughout the development process. Ensuring consistent relationships between various system models is an important part of an integrated design methodology. This paper describes an architectural approach to reasoning about relations between heterogeneous system models. The run-time base architecture of the system is used as a unifying representation to compare the structure and semantics of the associated models. Each model is related to the base architecture through the abstraction of an *architectural view*, which captures structural and semantic correspondences between model elements and system entities. The use of the architectural view framework to relate system models from different domains is illustrated in the context of a quadrotor air vehicle.**

***Keywords*-architectural views; multi-domain modeling; cyber-physical systems; component-connector;**

I. Introduction

Today's complex cyber-physical systems (CPSs) are created using models throughout the system development process, an approach referred to as model-based design (MBD) [7]. Models allow designers from different disciplines to develop and evaluate design alternatives within the context of formalisms relevant to selected aspects of the system. Each representation highlights certain features and occludes others to make analysis tractable and to focus on particular performance attributes. A particular modeling formalism typically represents either the cyber or the physical elements well, but not both. For example, differential equation models represent physical processes well, but do not represent naturally the details of computation or data communication. On the other hand, discrete formalisms such as process algebras and automata are well suited for representing concurrency and control flow in software, but are not particularly useful for modeling continuous phenomena in the physical world. Thus, the heterogeneity of elements in CPSs requires multiple perspectives and formalisms to explore the complete design space. Ensuring consistent relationships between various system models is an important part of the integrated MBD methodology.

We have developed the CPS architectural style as a system-level representation that is not prejudiced towards either the cyber or the physical side [2]. Architectures are annotated structural representations that describe systems at a high level of abstraction, allowing designers to determine appropriate assignment of functionality to elements, evaluate the compatibility of the parts, and make trade-offs between different quality attributes such as performance, reliability, and maintainability. This paper describes how the CPS architecture for a system provides a unified point of reference for multi-domain models based on heterogeneous formalisms. Our approach is to define relationships between system models at the architectural level, rather than developing a universal modeling language or a meta-modeling framework for translating between models from different formalisms. We believe that an architectural approach provides the right level of abstraction: one that captures the structure of and interdependencies in a system without attempting to comprehend all of the details of any particular modeling formalism.

The next section describes the use of the CPS architectural style to define base architectures for cyber-physical systems. The STARMAC quadrotor is introduced as a case study in this section. Section III introduces the concept of architectural views as means of relating heterogeneous models to a common base architecture. In Sect. IV, we illustrate the creation of three heterogeneous views in the context of the STARMAC quadrotor. Section V describes related work in this area, and the concluding section discusses ongoing work to extend our approach to represent and analyze multi-domain model consistency for CPSs.

II. A Unifying Architectural Representation

Architectures are often represented using a collection of architectural perspectives, which represent a set of related concerns [3]. The *component-and-connector (C&C) perspective* models a system as an annotated graph of components and connectors, in which the components represent principal computational and physical elements of a system's run-time structure, the connectors represent pathways of communication and physical coupling between components, and annotations represent properties of the elements [10]. In this work, the term 'architecture' is synonymous with a C&C architecture, because all the modeling formalisms of interest to us focus on analyzing properties and behavior of entities defined in the C&C architecture of the system under study. Based on this assumption, the CPS architectural style is also defined in the C&C perspective. Figure 1 illustrates the BA for the quadrotor.

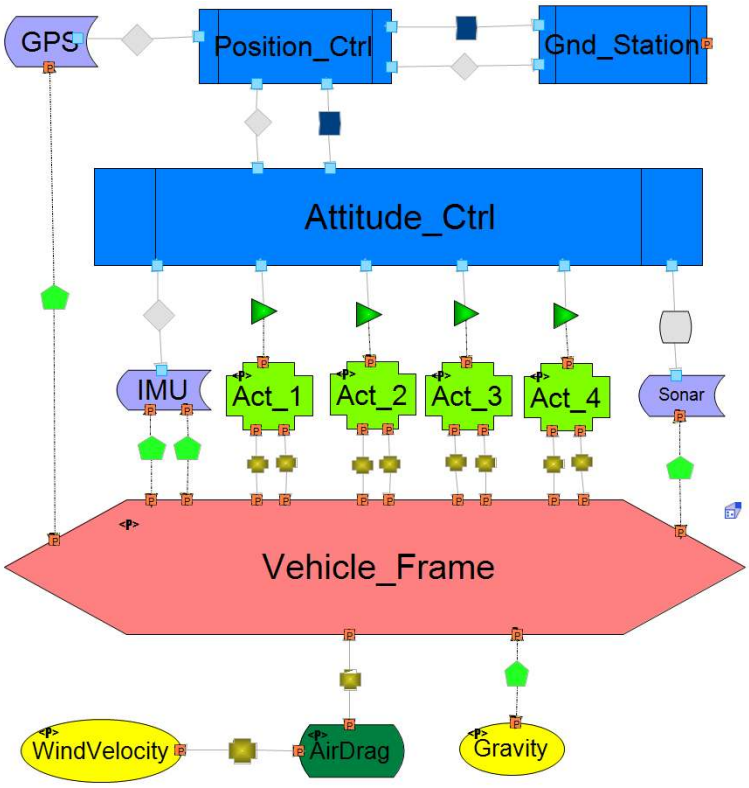

Fig. 1. Base Architecture of the STARMAC quadrotor.

This BA was created from the STARMAC implementation. The complete run-time architecture is modeled in the CPS style, which allowed us to represent both the cyber components (control algorithms and real-time software) and the physical dynamics (forces and torques imparted to the vehicle frame from physical sources). A more detailed description of the complete quadrotor CPS architecture is provided in [2].

Current C&C architectural styles, which focus primarily on software and computational infrastructures, are not comprehensive enough to describe a complete CPS. A CPS contains physical elements in addition to cyber entities, and includes elements representing interactions between these two domains. We have addressed this shortcoming through the development of a CPS architectural style [2] that augments traditional cyber architectures with elements corresponding to physical dynamics and laws. This architectural extension allows us to create a run-time representation of the complete CPS, called the system's base architecture (BA).

Definition 1. *The BA of a CPS is an instance of the CPS architecture style, which contains all the cyber and physical components and connectors that constitute the complete system at runtime.*

This definition implies that replicated functional units (for fault-tolerance) are also contained in the BA. The BA should contain enough detail to convey the nature of information and physical quantities flowing between components. In addition, the communication mechanism between components and relations between physical variables should be defined by the appropriate connectors. For new CPSs, the BA is built during the design phase from validated requirements and system specifications. For legacy CPSs, the BA is inferred from the implemented system, existing documentation and system models, and the knowledge of the system designers. In either case, we assume that the BA evolves as the design of the CPS evolves, throughout the system development lifecycle.

The following example of a real-time, embedded, multi-loop feedback system, will be used to illustrate the concepts of architectural views and their relation to the BA. The Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC) [6] is a quadrotor platform developed to test algorithms that enable autonomous operation of aerial vehicles. The aircraft has four rotors for actuation, arranged symmetrically about its body frame. The vehicle has an sensor suite consisting of an inertial measurement unit (IMU), a Global Positioning System (GPS) unit, and sonar. It implements a hierarchical control system, with a low-level attitude controller (AC) and a high-level position controller (PC). A remote ground station controller (GSC) generates reference trajectories for the quadrotor to follow, and has joysticks for control-augmented manual flight. The two onboard controllers communicate through a serial link. Communications between the PC and the GSC are managed over a WiFi network, using the UDP protocol.

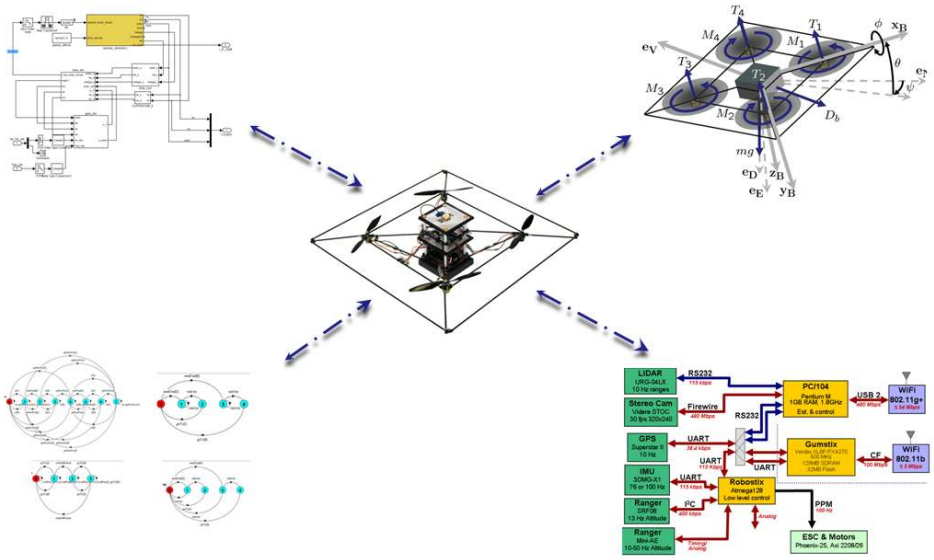

Fig. 2. How do we relate multi-domain models of a CPS?

III. Architectural Views

A CPS is typically described and analysed using multi-domain models, where each model focuses on a fixed set of concerns about the underlying system. Figure 2 shows four models of the STARMAC quadrotor that represent the same system from the physical, control design, software, and hardware domain perspectives.

In virtually all analysis tools, such models are constructed as collections of interacting components or modules. Thus, each model has a structure that can be viewed as an architecture with syntax and semantics defined by the particular formalism underlying the design of the tool. We would like to be able to define consistent relations between these models at some level of abstraction.

The approach proposed here focuses specifically on architectural views which represent the architectures of system models as abstractions and refinements of the underlying shared BA. In this context, well-defined mappings between a view and the BA can be used as the basis for identifying and managing the dependencies among the various models and to evaluate mutually constraining design choices. The BA thus becomes the repository for retaining results from various analyses and designs so that the interdependencies are explicit. This gives us the ability to reason about relations between models by studying their individual mappings to the BA of the system.

The relationships between elements in a model and entities in the BA will not generally be one-to-one, however. Current tools do not provide insights into the relationships between such heterogeneous models of a CPS. This represents a problem for architectural modeling, since it is generally impossible to understand how design decisions or analyses in one view impact those of another. From a structural perspective, an architectural view supports the description of a derived architectural model to abstract over details that are irrelevant for a particular analysis. The following definition formalizes the concepts of the BA and architectural views that we have described informally thus far.

Definition 2. *An architectural view $\mathcal{V}$ for a modeling formalism $\mathcal{M}$ is a tuple $< \mathcal{C}_V, \mathcal{R}_V^M, \mathcal{R}_{BA}^V >$ where:*

- *$\mathcal{C}_V$ is the component-connector configuration of the view, with the types, semantics, and constraints defined by the modeling formalism of the view*
- *$\mathcal{R}_V^M$ is a relation that associates elements in the model with elements in $\mathcal{C}_V$*
- *$\mathcal{R}_{BA}^V$ is a relation that associates elements in $\mathcal{C}_V$ with elements in the BA*

Figure 3 shows the conceptual relationship between system models, views, and the BA, based on definitions 1 and 2.

$\mathcal{R}_V^M$ is either one-to-one or an encapsulation of model entities, as defined by the modeler's choice of grouping. It effectively creates a "componentized" version of the model and allows grouping of multiple elements in the model to a single element in the view. $\mathcal{R}_{BA}^V$ is an encapsulation/refinement relation, which enables the system architect to group specific components and connectors in the view and map them to subparts of the BA. Some correspondences are declared explicitly by the architect while other correspondences are inferred, based on the semantics of the underlying view formalism.

One-to-many (encapsulation) and many-to-one (refinement) maps are allowed. However, many-to-many maps are not allowed since this can lead to inconsistent connections being hidden inside the encapsulated components.The component-connector structures resulting from carrying out element encapsulations on a view and on a BA are called an *encap-view*

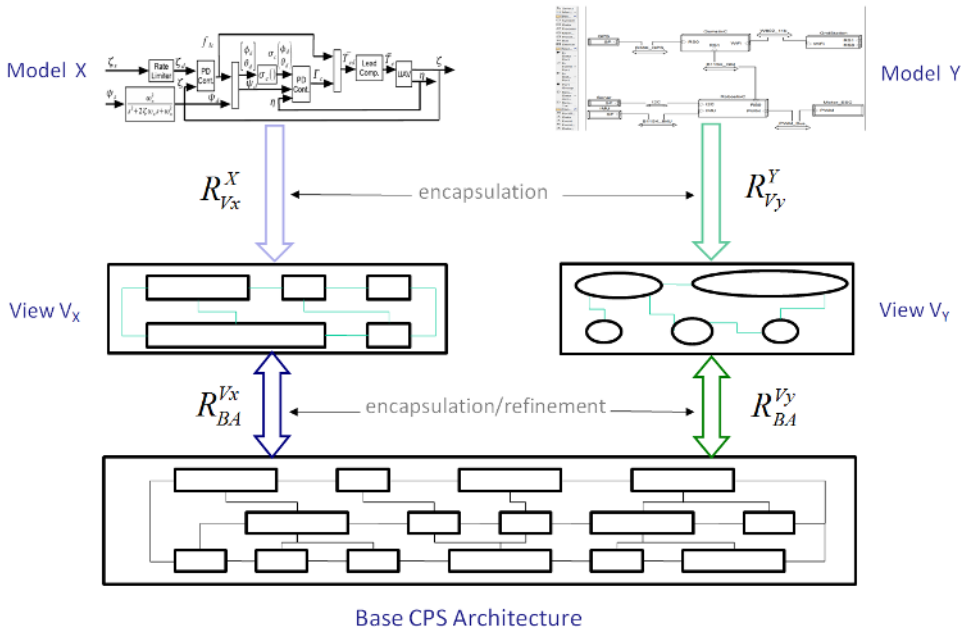

Fig. 3. Relationship between models and the BA through views.

and an *encap-BA*, respectively.

IV. Architectural Views of the Quadrotor

This section describes how heterogeneous models of the quadrotor can be related to the BA through architectural views. The choice of the modeling domains is motivated by the analysis and verification activities typically found in the design process of embedded control systems. In this case, the STARMAC design team had documented the software subsystems and the hardware architecture of the vehicle. We modeled the quadrotor physical dynamics in MapleSim from first principles, as well as studying the vehicle dynamics from existing control system models in Simulink.

A. Control View

From a control engineer's perspective, the quadrotor system can be viewed as a signal flow (Simulink) model. The position and attitude controller components in the BA are represented by the robostix and gumstix subsystems in the Simulink model. The vehicle dynamics are represented by the starmac_dynamics block, and the GPS and IMU sensors are defined by the Superstar_II and the 3DM blocks, respectively. Figure 4 illustrates the creation of the control view from a Simulink model. The relation $\mathcal{R}_V^M$ maps each top-level Simulink block to a component, and each group of signal lines between them to a connector, resulting in the control view's $\mathcal{C}_V$. The semantics for the $\mathcal{C}_V$ are derived from the underlying signal flow semantics of the Simulink metamodel.

For example, every connector in the control view represents a (cyber or physical) signal. Hence, semantically equivalent connectors between two components in the BA can be mapped to a single connector in the control view under this particular $\mathcal{R}_{BA}^V$. The mapping of four cyber-physical (C-P) connectors between the attitude controller and an encapsulated component (containing VehicleFrame) in the BA to a single connector between Robostix and Starmac in the view is shown in Fig. 5.

We disallow many-to-many maps between macro elements because the following type of situation could arise. Suppose that the architect decided to group the Robostix, Gumstix, and GPS components of the control view as one macro element. The architect also groups the position controller, attitude controller, and GPS components in the BA into a single element, and associates it with the macro element in the view. An inconsistent connector existing between Robostix and GPS (highlighted in Fig. 5) will be hidden away in the macro view element, and will not be detected if the control view and BA are compared for some type of consistency check.

B. Process Algebra View

Finite State Process (FSP) [8] is a process algebra where behavior is modeled in terms of event patterns, called processes that denote sets of event traces. Each event in a trace represents a discrete transition of a system. In general, FSP captures the behavior of cyber elements fairly well, while physical elements are described by abstracting away their continuous dynamics. The components in an FSP view are those entities whose behavior can be described by an FSP primitive process. A connector between two FSP components signifies that the two processes interact with each other through events and describes the protocol for that interaction, again as an FSP process.

The FSP specification of the quadrotor currently abstracts over the dynamics of the quadrotor and focuses on the communication between the ground station and position con-

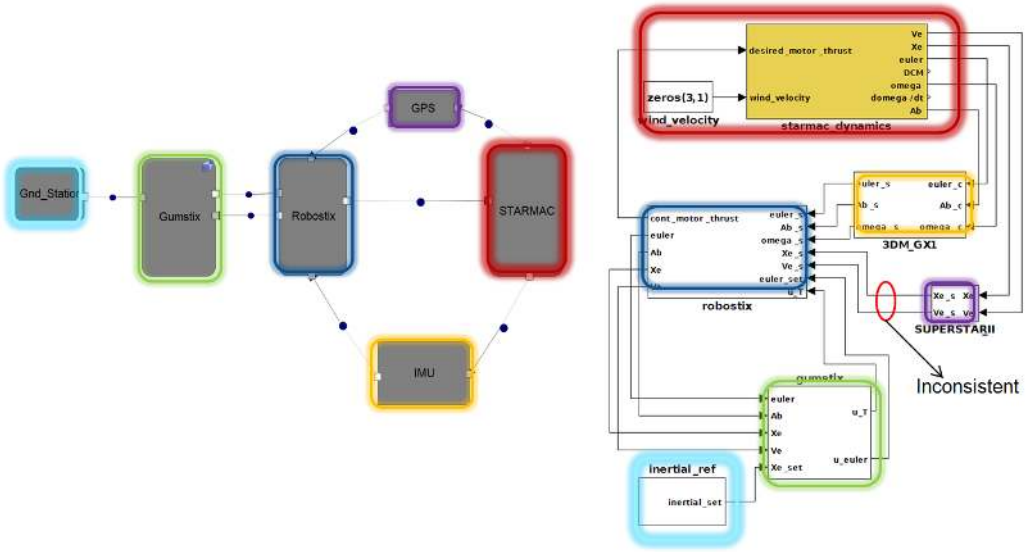

Fig. 4. Creating the control view from a Simulink model.

troller. The process algebra view is created by mapping each view entity to an FSP process in the specification, as shown in Fig. 6. The Gnd_Station component is mapped to the GroundStation process, which specifies how the GSC sends setpoints to the PC. The QuadRotor component is mapped to PositionController FSP process that describes how the ideal closed-loop quadrotor responds to position setpoints. The connector between Gnd_Station and QuadRotor is mapped to an FSP process that specifies the communication protocol between the two. The connector can be one of two types: a lossy connector represents a wireless UDP link while a lossless connector with retry models a wireless TCP. Having alternative connector protocols allows us to compare the behavior of the overall system depending on the protocol of the connection.

The mapping between the process algebra view and the BA is shown in Fig.7. The abstraction of vehicle dynamics is represented in the encapsulation of all the physical components in the BA into a single *QuadRotor* component in the view.

C. Physical View

The physical view models the dynamics of the vehicle in terms of the forces and torques applied by the rotors to the vehicle frame. The nonlinear dynamics of the quadrotor helicopter are those of a point mass m with moment of inertia $I_b \in R^{3\times3}$, location $\rho \in R^3$ in inertial space, and angular velocity $\omega \in R^3$ in the body frame. The vehicle undergoes forces $F \in R^3$ in the inertial frame and moments $M \in R^3$ in the body frame, yielding the equations of motion,

$$\vec{F} = -D_B\vec{e}_V + mg\vec{e}_D + \sum_{i=1}^{4} T_i\vec{z}_B$$

$$\vec{M} = \sum_{i=1}^{4} T_i(\vec{r}_i \times \vec{z}_B)$$

where D_B is the aerodynamic drag force, and g is the acceleration due to gravity. $R_{R_j,I}$ and $R_{R_j,B}$ are the rotation matrices from the plane of rotor j to the inertial coordinates and the body coordinates, respectively.

The CPS style enables the formal representation of such dynamic behavior in the overall system architecture. The dynamics model of the quadrotor is implemented in the Modelica language, and the semantics of the $\mathcal{C}_V$ are defined in terms of non-causal interconnections between the effort and flow variables of each attached component's ports.

The mapping between the physical view and the BA is shown in Fig. 9. The set of view components map to a subset of the elements in the BA.

V. Related Work

Multiple efforts have focused on supporting multi-view, model-based system development. The SAE AADL (Architecture Analysis and Design Language) is an international standard for predictable model-based engineering of real-time and embedded computer systems [5]. AADL offers a set of predefined component categories to represent real-time systems and it is capable of describing functional component interfaces like data and control flows, as well as non-functional aspects of components like timing properties. However, AADL does not support architectural representation of physical domain entities (except as generic 'device' components), nor does it address how heterogeneous views can reconciled.

Ptolemy II is a tool that enables the hierarchical integration of multiple "models of computation" in a single system, based on an actor-oriented design [1]. Even though Ptolemy II supports hierarchy and incorporation of multiple formalisms at the detailed simulation level, it is not possible to define architectural styles or high-level design tradeoffs. In addition, there is no support for acausal, equation-based modeling of

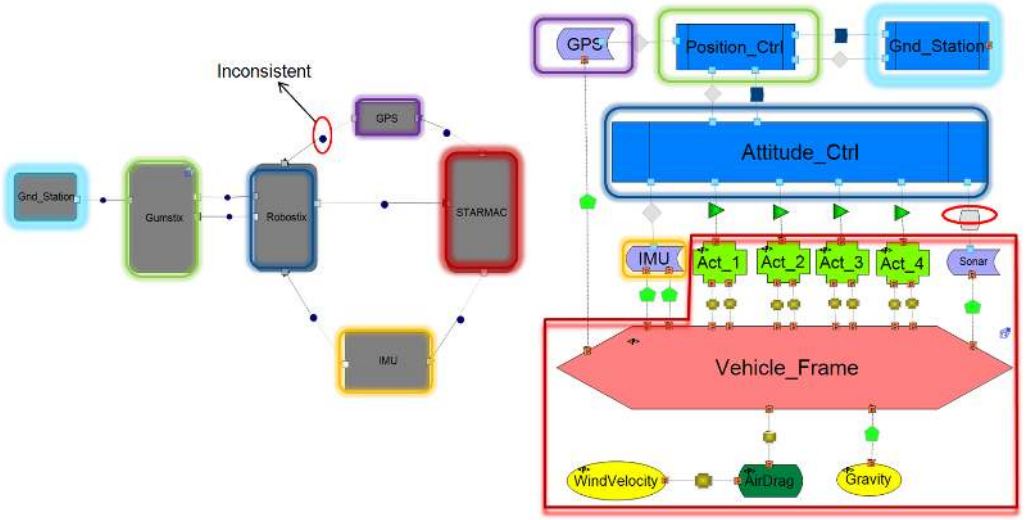

Fig. 5. Mapping between control view and BA.

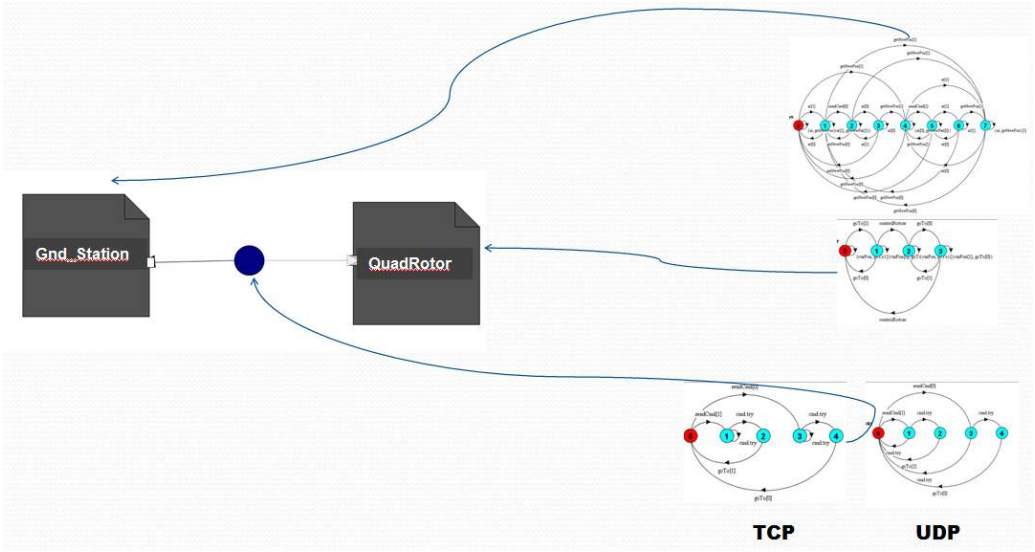

Fig. 6. Creating the process algebra view from an FSP specification.

physical systems, since the underlying formalism is event-based communication.

The Vanderbilt model-based prototyping toolchain provides an integrated framework for embedded control system design [9]. It provides support for multiple views, such as functional Simulink/Stateflow models, software architecture, and hardware platform modeling along with deployment. The toolchain's ESMoL language has a time-triggered semantics, which restricts the functional view to Simulink blocks that can only execute periodically. There is currently no support for additional views (e.g., physical or verification models), nor a notion of consistency between additional system views. In contrast, our work focuses on architecture-level view comparison, not on meta-modeling or model transformations.

SysWeaver [4] is a model-based development tool that includes a flexible code generation scheme for distributed real-time systems. The functional aspects of the system are specified in Simulink and translated into a SysWeaver model to be enhanced with timing information, the target hardware model and its communication dependencies. The translation from Simulink is not completely automated if closed-loop controllers are present. Sysweaver's computational framework semantics is restricted to tasks that exchange information via message-passing (time or event-based). There is also no support in SysWeaver for a physical plant modeling view.

VI. Discussion

Once we have the ability to relate heterogeneous system models to the BA through the mechanism of architectural views, several interesting questions can be asked. A natural

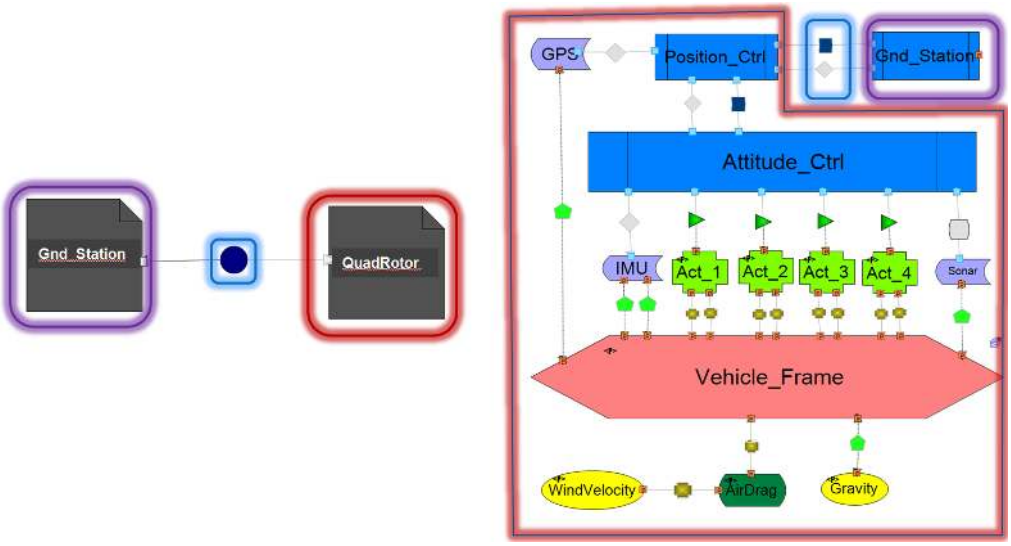

Fig. 7. Mapping between process algebra view and BA.

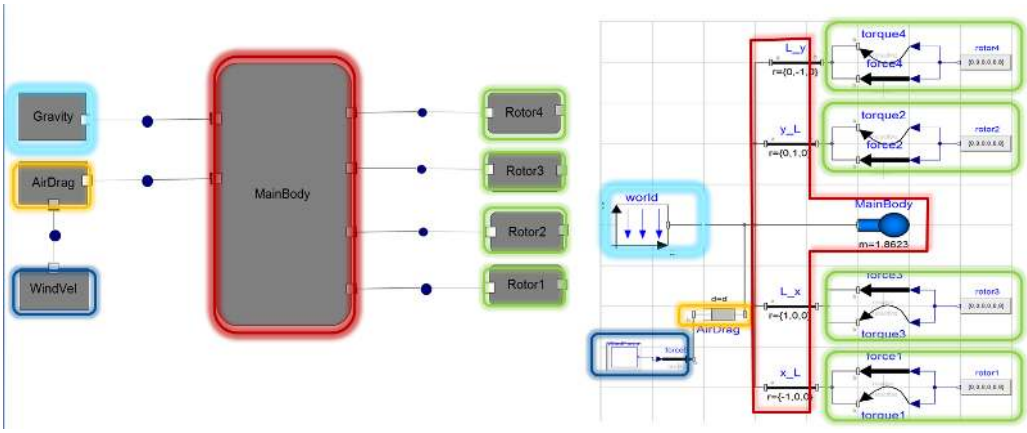

Fig. 8. Creating the physical view from a Modelica model.

one is: what does it mean for each system model to be consistent (in some sense) with the underlying system representation?

Consistency can be studied between a single model and the underlying system or between multiple models of the same system. Consistency of a single model with the architecture makes it possible to relate (and subsequently use) verification results derived from the model to the final system implementation. This is possible if the final run-time system is generated in a systematic way to guarantee conformance to the architecture. If multiple models describe the same system, then the models should be based on consistent assumptions about the system's parts, including the parts that are abstracted away. Only then can the different sub-systems designed using these models be integrated, and the final composed system behavior be the same as the behavior expected using the individual model analysis results.

Consistency can also give the designer the ability to relate system requirements to its models. The BA is assumed to be constructed from validated stakeholder/system requirements. Hence, the BA contains only components and connectors that can be traced back to particular requirements. By enforcing that each view maintain consistency with the BA, we obtain a way to carry out requirements traceability for the corresponding model as well. Hence, the model cannot contain extraneous elements (or connections between elements) that are not mandated by some system-level requirement. This gives the modeler a mechanism to verify whether the model complies with the decisions (and future changes) made at the system architecture level are reflected in each model.

The exploration of these issues form the next steps in our approach to multi-domain modeling using architectural views.

ACKNOWLEDGMENT

This work is supported in part by National Science Foundation (NSF) under grant no. CNS0834701 and by Air Force

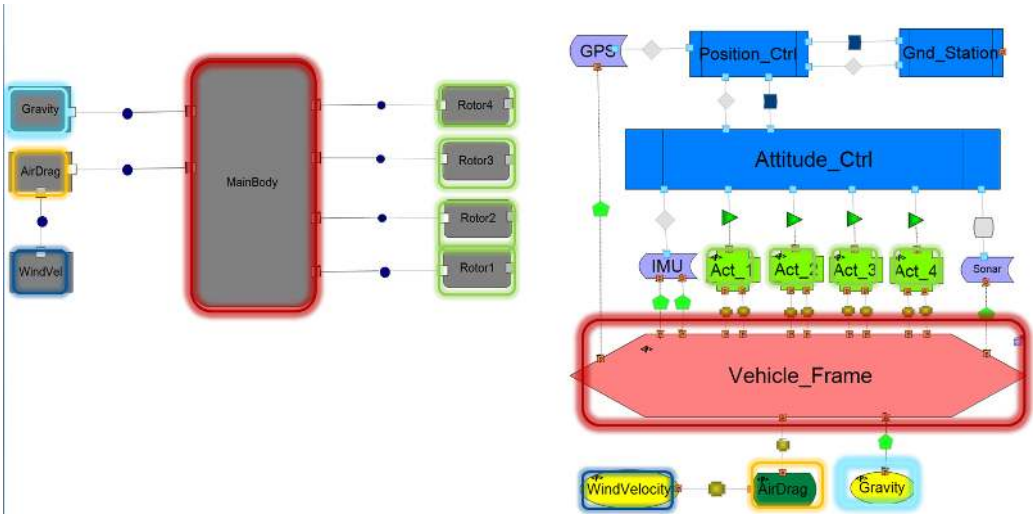

Fig. 9. Mapping between physical view and BA.

Office of Scientific Research (AFOSR) under contract no. FA9550-06-1-0312.

REFERENCES

[1] S. S. Bhattacharyya, E. Cheong, and I. Davis. Ptolemy II heterogeneous concurrent modeling and design in java. Technical report, 2003.

[2] A. Bhave, D. Garlan, B. Krogh, A. Rajhans, and B.Schmerl. Augmenting software architectures with physical components. In *Proc. of the Embedded Real Time Software and Systems Conf. (ERTS^2 2010)*, 19-21 May 2010.

[3] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2002.

[4] D. de Niz, G. Bhatia, and R. Rajkumar. Model-based development of embedded systems: The Sysweaver approach. *IEEE Real Time Technology and Applications Symposium*, pages 231–242, 2006.

[5] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The architecture analysis and design language (aadl): An introduction. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, Feb 2006.

[6] G. Hoffman, S. Waslander, and C. Tomlin. Quadrotor helicopter trajectory tracking control. In *Proc. of the AIAA Guidance, Navigation, and Control Conference*, 2008.

[7] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing domain-specific design environments. *Computer*, 34(11):4451, 2001. doi:http://dx.doi.org/10.1109/2.963443.

[8] J. Magee and J. Kramer. *Concurrency: State Models and Java Programming, Second Edition*. Wiley, 2006.

[9] J. Porter, P. Volgyesi, N.Kottenstette, H.Nine, G.Karsai, and J. Sztipanovits. An experimental model-based rapid prototyping environment for high-confidence embedded software. In *RSP '09: Proceedings of the 2009 IEEE/IFIP International Symposium on Rapid System Prototyping*, pages 3–10, Washington, DC, USA, 2009. IEEE Computer Society.

[10] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.