Platform-Independent Autonomy Modeling

Oleg Sokolsky and George Pappas
Department of Computer and Information Science
University of Pennsylvania
3330 Walnut St.
Philadelphia, PA 19104

***Abstract*— We describe an approach for high-level modeling behaviors of autonomous vehicles and an infrastructure for executing these behaviors on a particular vehicle platform. The language directly represents behavioral primitives and constraints on their composition. The control infrastructure maps these behavioral primitives on the native vehicle interface in a model-driven fashion. As a result, the user is presented with an abstract motion planning interface that hides the intricate details of the vehicle implementation.**

I. Introduction

Autonomous vehicles are bound to play a more and more important roles in our lives. Already, unmanned aircraft are employed by the army to perform reconnaissance missions and look for explosives. Researchers use robots to explore narrow shafts in ancient Egyptian pyramids. Control of autonomous vehicles and vehicle formations is an important research area. A number of research projects, for example [9], consider intelligent highways, where cars can be driven autonomously, the use of robots in emergency response scenarios, etc. A variety of robotic platforms, both ground and aerial, that can be used in autonomous navigation are available on the market. Among them one can find fixed wing aircraft and helicopters; wheeled and tracked ground vehicles; even legged robots.

Autonomous vehicles generally offer an interface that allows them to be remotely controlled either by a human operator or a computer-controlled motion planner. The degree of autonomy that these vehicles offer varies greatly. Most are capable of following a trajectory laid out by way points, while others may in addition have a built-in capability to avoid obstacles.

In this paper, we describe a research effort recently initiated at the University of Pennsylvania to provide platform-independent means to control autonomous vehicles and their formations based on high-level models. The effort builds upon two kinds of expertise. One the one hand, we have studied control of autonomous vehicles and their formations for a long time [3], [5], [13]. This work has been done from the control-theoretic and experimental perspective, without a significant modeling component. However, this experience allowed us to identify requirements for the modeling language and infrastructure. On the other hand, two long-standing lines of modeling research concerned hierarchical hybrid systems [1], [2] and resource constraint modeling [8], [11].

The overall structure of the approach is shown in Figure 1. At design time, the framework is populated with the available vehicles. Models of vehicles are entered and stored in the database. Before a vehicle or formation is launched and navigated, the database is queried and capabilities of the vehicle, in terms of the available motion primitives, are presented to the navigation component (which may be an automatic controller or a human user). Using these components as building blocks, the navigation component constructs the desired behavior of the vehicle at run time. Internally, the high-level behavioral primitives chosen by the navigation component are automatically translated into the commands of the native vehicle control interface and sent to the vehicle. Sensory data obtained from the vehicle go through the reverse process.

The framework thus has two major components that are described in more detail in the rest of the paper. Section II presents the language for modeling autonomous behaviors. Section III presents the design and execution framework for supporting control of autonomous vehicles based on this modeling approach. Finally, Section V describes our plans for implementing the framework.

II. Modeling language for vehicle autonomy

In order to formally describe diverse autonomous vehicles, with various degrees of autonomy and capabilities, we need to have a language, that can express motion primitive and offer composition rules that allow to construct more complex motion primitives from simpler ones. To increase the expressive power of the language, we are aiming to go beyond motion primitives and capture other aspects of a vehicle such as vehicle resource constraints, payload constraints, dynamic constraints, as well as all autonomy, sensor, collective and failure behaviors in a modular fashion. Every vehicle is assembled from a collection of modes that describe individual behaviors of the model, a collection of elementary resources such as fuel and payload, and a collection of sensors that are deployed on the vehicle. Operationally, modes require access to resources with certain parameters (for example, fuel consumption rate) in order for the mode behaviors to be performed.

Every behavioral mode or primitive has associated pre-conditions (when entering the behavior) and possibly non-deterministic post-conditions (when exiting the behavior). In addition to the array of behavioral (or capability) primitives, the language is equipped with natural notions of composition that will allow, among others, the description of sensor-based autonomous behaviors (by composing a sensor primitive with an autonomy primitive of a vehicle), collective team behaviors (by composing together different vehicles), or collective

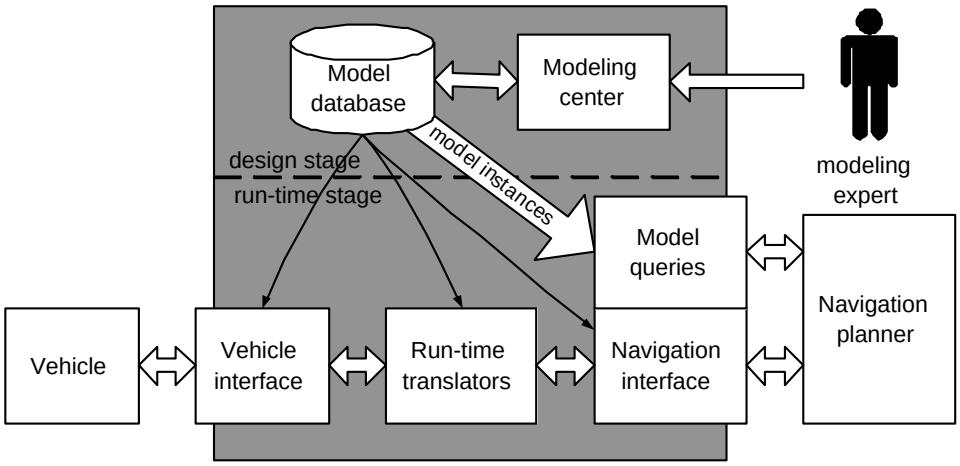

Fig. 1. Architecture of the navigation infrastructure

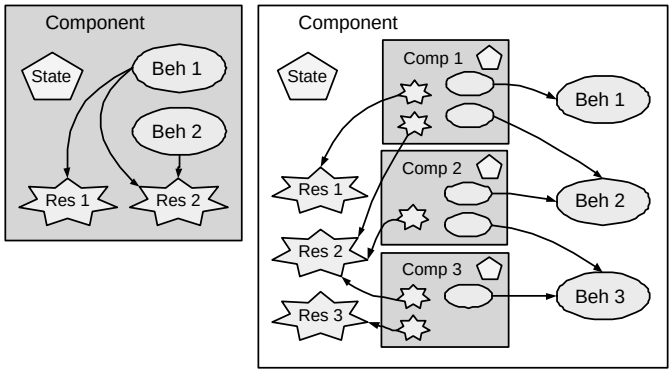

Fig. 2. Agent structure and composition

sensing behaviors (by composing sensors). The modularity of the language and the compositional semantics not only results in a highly expressive language but also provides a very efficient mechanism for representing platform behaviors at different levels of autonomy. An important requirement for an autonomy modeling language is that it is extensible. New vehicles supporting different autonomy primitives become available and should be easily incorporated into the modeling framework. The structure of our language can naturally accommodate new behavioral primitives, new vehicles and sensors with higher levels of autonomy, additional failure capabilities, or next-generation support for sophisticated collective behaviors. This will allow our modeling framework to evolve with technology developments.

The main building block of the language is an *agent*, which has an interface capturing agent behaviors and agent resources. We distinguish several types of agents. Agents of type `vehicle` capture motion primitives of a platform. Agents of type `sensor` capture sensing primitives. Composing a vehicle agent together with a number of sensor agents we obtain a composite agent of type `platform`. Composing several platforms together we can get a `team` agent. At run time, agents encapsulate state information. Agents can be put together to form more complex agents. Agents and their composition are illustrated in Figure 2. Each agent offers a collection of behaviors that it can engage in, along with its resources and state information. When agents are composed together, their behaviors are composed concurrently to form behaviors of the composite agent, and resources and state of the agents are joined to form resources and states of the composite agent. Note that not all agents have to participate in every composite behavior and that agents may have shared resources and state data.

Composition rules of the language include two kinds of composition: agent composition and behavior composition. Agents are always composed in parallel. Rules of agent composition ensure type compatibility. For example, only agents of platform type can be used to produce a team agent, and a platform agent should have exactly one vehicle agent in it. When two agents are composed together, their behaviors are composed using behavior composition rules. Behaviors can be composed in parallel or sequentially. Behavior composition rules are based on pre- and post-condition constraints that are associated with each behavior. The notion of behavior composition that we employ is not unlike the approach used in hybrid systems modeling, but has several important distinctions. In hybrid systems formalisms such as CHARON [1], simple behaviors are given by differential and algebraic constraints, and such behaviors are sequentially composed by means of discrete switches that happen under certain conditions. State machines formed in this way can then be composed concurrently, usually in an unconstrained way. In our approach, instead of specifying transitions, we are giving constraints that restrict switching and allow the navigation to effect switches between any behaviors whenever constraints are satisfied. Parallel composition is also subject to constraints imposed by the resources in the model.

A. Constraints and resources

The basis for composition of behaviors is formed by pre- and post-conditions that are associated with each behavior. Pre-condition constraints are defined in terms of the state of

the model and availability of resources that are necessary for a behavior to proceed. Post-conditions express the effect of performing a behavior on the state and resources of the agent. For composing behaviors in parallel, their pre-conditions have to be simultaneously satisfied. For sequential composition, the post-condition of the first behavior must satisfy the pre-condition of the second behavior.

We distinguish two kinds of resources that are necessary for a behavior to be feasible. These are serially reusable and consumable resources. A serially reusable resource is can be used by only one behavior at a time. Once a resource is released, another behavior can use it. Serially reusable resources are a uniform way to capture incompatible behaviors. As an example, consider a camera that can point down to observe objects on the ground, or point sideways to peek into windows. Clearly, however, one camera cannot do both simultaneously. A generalization of a serially reusable resource is a multi-capacity resource, which can be used by multiple behaviors simultaneously, as long as total usage does not exceed the resource capacity. An example of such resource is a power battery. Several agents may draw on the same battery. However, the battery can provide a limited amount of current, which limits the number of agents that rely on it. Consumable resources, by contrast, limit sequential composition of behaviors, or the duration of a particular behavior. Using the same battery as an example, the total amount of power that the battery has is a consumable resource that gets depleted as the behavior proceeds.

Constraints are, mathematically, predicates over the model state. Evaluation of constraints can be done symbolically during model construction, or using concrete values at run time. Symbolic evaluation of constrains is used in deciding which behaviors are compatible. Specifically, if a constraint involves a serially reusable resource, we can easily decide when two behaviors use the same resource and disallow their parallel composition. When pre-conditions of two behaviors cannot be fully resolved symbolically, the conjunction of the pre-conditions is the pre-condition for the composite behavior.

B. Modeling autonomous behaviors

Behaviors that capture navigation primitives of vehicles are usually defined over three-dimensional trajectories. All vehicle agents, therefore, contain 3D points as their state. Additional state information captures the state of consumable resources. As an example, we consider modeling of a hypothetical UAV airframe inspired by the UAV platform we have at Penn. The vehicle agent is shown in Figure 3. The model specifies parameters of the vehicle, model state that needs to be kept at run-time, resources that the model has, and, finally, behaviors of the model. Keywords of the language are highlighted in bold face.

The model in Figure 3 specifies that the vehicle `PennUAV` has its position and orientation as the state and fuel as its consumable resource with the total capacity fuelTankCapacity. At run time, remaining capacity of a resource represent additional state information. Behaviors of the vehicle capture the different flight modes available for the vehicle. One of the behaviors is a take-off mode, which can be invoked when the vehicle is on the ground. Other behaviors include cruising at a specific speed with fixed orientation, turning to a specific orientation, and straight-line navigation to a specified absolute or relative position. Behaviors put constraints on the evolution of state during a behavior, which in this case are specified by differential inclusions. Note especially that the vehicle offers waypoint navigation that is defined as a sequential composition of single-point behaviors. Switching of primitive behaviors in this sequential composition is controlled by constraints that specify that the next waypoint has been reached before switching to the next behavior. Such sequential composition allows us to represent different levels of autonomy within the same model. The navigation component will be able to use less autonomous behaviors in addition to more autonomous behaviors. We also establish a partial order on autonomous behaviors to reflect that more autonomous behaviors have a higher preference and should be used when there is a choice between equally viable solutions.

Pre- and post-conditions of a behavior specify how this behavior can be composed with other behaviors. In the vehicle model above, behavior `takeOff` can happen when the vehicle is on the ground, and once the behavior completes, the target height has been reached. The precondition for the navigation behaviors is that the vehicle is already flying. Therefore, a navigation behavior can not occur simultaneously with take off since their pre-conditions cannot both be satisfied in any state. However, various navigation behaviors, for example, the one that specifies the speed and the one that specifies the angular direction, can be composed together since they have the same precondition and do not share conflicting resources.

C. Modeling sensor behaviors

Simple vehicle sensors that provide scalar data such as vehicle speed, orientation, fuel level, etc., are best modeled as free variables of the model. These variables can be used in the constraints on the model behaviors, as well as made available to the navigation component for use in trajectory planning. However, complex sensors such as cameras and GPS, which have non-trivial behaviors and multiple operation modes, require special modeling facilities to adequately capture their behaviors. For example, a camera may have a single-frame mode, when it takes a picture upon receiving a command, and a continuous mode, parameterized by the capture rate. In addition, it may be oriented to point in different directions. Depending on the camera, it may or may not be able to take pictures during re-orientation. Camera control systems may also interact with the GPS sensor on the vehicle and be instructed to take a picture when a specific coordinate is reached. The modeling language has to be able to capture all of these aspects of sensor operation and use then in model composition. The model of each sensor should include a description of:

- Ability to detect objects of interest (for example, such targets as ground vehicles).

```
vehicle pennUAV {
   parameters
      wingSpan:distance; maxAirSpeed:speed; fuelTankCapacity:gallons;
      maximumOperationalRange:range; maximumOperationalAltitude:distance
   state
      pos:location; theta:orientation
   resources
      consumable fuel limit MaxFuelCapacity
   behaviors
      takeOff( height:altitude )
         pre pos.z = 0; post pos.z = height
         diff pos.x = [speed_min, speed_max];
         diff pos.y = [speed_min, speed_max]
         diff pos.z = [climbRate_min, climbRate_max]
         diff fuel = [-maxBurnRate, -minBurnRate]
      cruise( v:speed )
         pre pos.z > 0
         diff pos.x = [xdot_min, xdot_max]
         diff pos.y = [ydot_min, ydot_max]
         diff fuel = [-maxEconRate, -minEconRate]
      turn( heading:orientation )
         pre pos.z > 0; direction = sign(heading-theta)
         diff theta = [turnRate_min*direction, turnRate_max*direction]
         diff fuel = [-maxEconRate, -minEconRate]
      goToAbs( v:speed, target:location )
         pre pos.z > 0; post pos = within ( pos, tolerance)
         diff pos.x = [xdot_min, xdot_max]
         diff pos.y = [ydot_min, ydot_max]
         diff pos.z = [zdot_min, zdot_max]
         diff fuel = [-maxEconRate, -minEconRate]
      goToRel( v:speed, relTarget:point)
         pre pos.z > 0; post pos = within ( pos, tolerance)
         diff pos.x = [xdot_min, xdot_max]
         diff pos.y = [ydot_min, ydot_max]
         diff pos.z = [zdot_min, zdot_max]
         diff fuel = [-maxEconRate, -minEconRate]
      navigate( v:speed, wp:pointList )
         seq goToRel(v, wp(i)) { pos == wp(i+1) }
}
```

Fig. 3. A model of an autonomous vehicle platform and its behaviors

- Ability to classify objects or targets.
- Information provided by the sensor (for example, target bearing and range).
- Description of the quality of information (for example, uncertainty associated with the estimates).

The emphases of the sensor modeling is on the functional and behavioral aspects rather than on the physical attributes of the sensor. The physical attributes, such as focal length of the camera, the pixel size etc., are important since they essentially derive the functional and behavioral characteristics. The functional and behavior characteristics, however, are the subjects of interest high-level modeling. Sensors commonly found on autonomous vehicles can be broadly grouped into three categories: imaging sensors, unidimensional signal sensors, and discrete signal sensors. Examples of imaging sensors include color video cameras, low-light TV cameras, and infrared cameras. Examples of unidimensional signal sensors include acoustic sensors, and radars. A discrete signal sensor, for example, can be a laser ranging sensor or a GPS sensor. The interfaces of the notional sensor model for each class allow sensors to be controlled during navigation. Attributes and behaviors of a notional imaging sensor mode are identified in Table I.

Additional functional and behavioral characteristics that provide the most benefits to the HURT sensor modeling will be identified and modeled in the notional ontologies. The prototype ontologies will include some physical attributes of the sensor in order to derive the functional behaviors.

A model of a notional sensor platform is shown in Figure 4. Note that this sensor platform description does not explicitly model any properties that might be inferred (inherited) by mounting it on a vehicle platform. A model of a notional sensor will include as attributes the number of pixels in the horizontal and vertical direction, the maximum field of view in both directions. The abstract model will relate the number of pixels that need to be subtended by the target for detection and the number of pixels for classification. Note that advanced sensor features, for example built-in automatic target

Functional / Behavioral Characteristics	Description
Function	Target acquisition, situation awareness
Functional outputs	An image which is a 2D projection of a 3D scene, annotated targets, tracks.
Spectral domain	Color, intensity or infrared.
Control parameters	Pan, tilt, zoom
Field-of-View coverage	The reconnaissance and surveillance area
Viewing geometry	Azimuth, and aspect angle, height
Operable time of day	Day, night, dust etc.
Operable weather	Sunny, cloud, fog, shadows, snow
Operable range	Maximum and minimum distances

TABLE I

FUNCTIONAL AND BEHAVIORAL CHARACTERISTICS OF AN IMAGING SENSOR

recognition, which themselves possess complex behaviors, can be included as sub-agents of the sensor agent.

D. Internal vs. external composition

The composition rules of the language will allow us to construct complex behaviors from simpler ones. For example, composing a sensor behavior with an autonomy mode will naturally result in sensor-based autonomous behaviors. In another example, waypoint routing and obstacle avoidance can be primitive behaviors of a platform that can be used separately or, on some platforms, can be used together in a complex behavior that follows waypoints while avoiding obstacles. Rules of composition specify when complex behaviors can be formed. For example, the behavior above may be used only assuming that waypoints themselves are safe from obstacles by a certain margin. This composition can be performed on two levels. On the one hand, behaviors can be combined within the model, by the modeler, and a composite behaviors presented to the navigation component when the model is queried. We call this composition internal. On the other hand, external composition is performed by the navigation component at run-time.

Both internal and external composition are important for navigation. Internal navigation allows the navigation component to use the highest degree of autonomy afforded by the vehicle. Since details of composite behaviors are hidden, navigation becomes simpler. On the other hand, external composition may allow more precise control of the vehicle since composition is performed based on the run-time data and can be dynamically adjusted. Of course, more computation is required to navigate the vehicle in this case.

E. Failure behaviors

Capturing failure modes in our behavior-based modeling language is very natural as failures could be modeled as restricting the available nominal autonomy or sensor modes. In addition to this binary fault model, explicit failure behaviors could be modeled as additional primitives, resulting in perhaps dramatic and structural changes in the nature of the constraints. For example, an actuator could be inactive or stuck at a particular level leading to structurally different dynamic behavior for the platform. From the navigation perspective, a failure will result in a change in the set of behaviors that can be used in constructing plans. In addition to capturing nominal and failure behaviors, also want to support the modeling of fault-handling autonomy or reconfiguration-logic that may be built into the vehicle controller.

III. MODEL-DRIVEN AUTONOMY CONTROL FRAMEWORK

High-level behavior-oriented models allow us to have a uniform interface that is offered to the navigation component. As illustrated by Figure 1, the interface is used in two phases. The set-up phase allows the navigation component to explore the capabilities of the available vehicles. The capabilities are presented as available behaviors and constraints on their use. Then, in the run-time phase, the navigation component sends navigation commands to the vehicles by invoking behaviors offered by each vehicle, according to the constraints of the model.

A. Run-time navigation adapters

Since behaviors requested by the navigation component are specified at the level of the abstract model, at run time these behaviors have to be transformed into commands in the native interface of the vehicle. By the same token, sensory data provided by the vehicle has to be collected using the native interface and delivered to the navigation component in an abstract form. Components that perform this run-time translation are called navigation adapters. An adapter is defined for each vehicle and each model that can be applied to this vehicle. We are exploring the use of meta-modeling techniques [12], [10] for automatic generation of adapters.

IV. RELATED WORK

There has been much work on languages for autonomous behaviors. We mention here the motion description language introduced in [4], maneuver automata [6], and motion graphs [7], among many others. The major difference between their approaches and ours is that motion languages are used in navigation planning. That is, the navigation component in our architecture may well be using one of these languages to come up with a sequence of behaviors needed to achieve a certain goal. The primary intent of our language is to provide an abstract interface for the navigation component that hides the details of the platform interface, simplifying the control task.

V. DEVELOPMENT PLANS AND CONCLUSIONS

The project described in this paper is currently in a very early stage. We are taking a step-by-step approach, building on our existing experience and gradually expanding it to encompass new aspects. The primary goal of the first stage of the project is to create a subset of the language that lets us interface with the vehicles we currently have in the GRASP lab. These vehicles include a one-quarter scale Piper Cub J3 airframe, a blimp, a number of wheeled carts equipped with omni-directional cameras and GPS sensors, and Sony

```
sensor EO {
   parameters
      pixels pixelsHorizontal, pixelsVertical;
      fieldOfView maxFOVhorizontal
   state
      isrState
      ContactType
   resources
      consumable battery limit batteryCapacity
   output
      ContactType, Bearing, aBearingError, Range, RangeError,rawData,
      detectionQuality, ...
   behaviors
      activeTarget(Bearing, BearingError, Range, RangeError)
      passive(GridGranularity, TemporalGranularity, GridType, rawDataMatrix)
      scanningMode(ContactType, Bearing, BearingError, Range, RangeError)
   agent ATRprocessor {
      ...
   }
}
```

Fig. 4. A model of a notional sensor platform

AiBO robot dogs. An important criterion for success is to be able to control these vehicles via the abstract interface as efficiently as we can by existing controllers, specially crafted to use native interfaces of the vehicles. At this stage, we will manually construct the navigation adapters, using parts of the existing controllers. From there, we will expand our work in two directions. One direction is to accommodate more vehicles into the framework, covering the full scope of diversity in vehicle types and capabilities. The second direction is towards automatic generation of navigation adapters.

To summarize, we believe that customizing hybrid systems and resource-oriented modeling to the domain of autonomous vehicles will allow us to develop an efficient abstract interface for the uniform control of diverse vehicles with varying degrees of autonomy. We hope to demonstrate that concentrating on a narrow domain will allow to more fully utilize the potential of model-driven techniques.

Acknowledgments. The authors wish to thank Vijay Kumar and C.J. Taylor from the GRASP lab at Penn for fruitful discussions on sensor modeling. Research has been supported in part by NSF grants CCR-0086147 and CCR-0209024, and by ARO grant DAAD19-01-1-0473.

REFERENCES

[1] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 90(1):11–28, Jan. 2003.

[2] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional refinement for hierarchical hybrid systems. In *Proceedings of Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 33–48. Springer-Verlag, Mar. 2001.

[3] S. Bayraktar, G. Fainekos, and G. J. Pappas. Experimental cooperative control of unmanned aerial vehicles. Submitted for publication.

[4] R. W. Brockett. Hybrid models for motion control systems. In H. Trentelman and J. C. Willems, editors, *Perspectives in Control*, pages 29–54. Birkh, 1993.

[5] R. Fierro, A. Das, J. Spletzer, J. Esposito, V. Kumar, J. Ostrowski, G. Pappas, C. Taylor, Y. Hur, R. Alur, I. Lee, G. Grudic, and B. Southall. A framework and architecture for multirobot coordination. *International Journal of Robotics Research*, 2002.

[6] E. Frazzoli, M. Dahleh, and E. Feron. A maneuver-based hybrid control architecture for autonomous vehicle motion planning. In *Software Enabled Control: Information Technology for Dynamical Systems*. IEEE Press, 2003.

[7] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of the 29^{t}h annual conference on computer graphics and interactive techniques*, pages 473–482, 2002.

[8] I. Lee, A. Philippou, and O. Sokolsky. A general resource framework for real-time systems. In *Workshop on Radical Innovations of Software and Systems Engineering in the Future*, number 2941 in LNCS, pages 234–248. Springer, Oct. 2002.

[9] A. Puri, P. Varaiya, and V. Borkar. Driving safely in smart cars. In *Proceedings of Hybrid Systems III*, number 1066 in LNCS, pages 362–376, 1996.

[10] K. Schloegel, D. Oglesby, E. Engstrom, and D. Bhatt. Composable code generation for model-based development. In *Proceedings of 7^{t}h Int'l Workshop on Software and Compilers for Embedded Systems*, 2003.

[11] O. Sokolsky, A. Philippou, I. Lee, and K. Christou. Modeling and analysis of power-aware systems. In *Proceedings of TACAS '03*, volume 2619 of *LNCS*, pages 409–425, Apr. 2003.

[12] J. Sztipanovits and G. Karsai. Model-integrated computing. *IEEE Computer*, 30(4):110–112, Apr. 1997.

[13] H. Tanner, G. J. Pappas, and V. Kumar. Leader to formation stability. *IEEE Transactions on Robotics and Automation*, 2004.