A Formal Approach to Reactive System Design: Unmanned Aerial Vehicle Flight Management System Design Example

T. John Koo, Bruno Sinopoli
Alberto Sangiovanni-Vincentelli, Shankar Sastry

Robotics and Intelligent Machines Laboratory
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, CA 94720
{koo,sinopoli,alberto,sastry}@eecs.berkeley.edu

Abstract

This paper presents a formal methodology for the design, implementation and validation of reactive systems. The methodology has been applied to the design of a Flight Management Systems (FMS) for a model helicopter in the BEAR project[9]. POLIS[2], a design tool developed at the University of California at Berkeley, is extensively used. The automation of the design problem and the validation techniques provided by this tool allow to shorten prototyping time and to prove the correctness of the properties of the system. Automatic code generation guarantees error free implementation, which is fundamental in safety critical applications. Simulation of the entire design is performed using Ptolemy, a hierarchical heterogeneous simulation environment.

1 Introduction

Reactive systems [7] react continuously to their environment at the speed of environment. Reactive systems are prominent in industrial process control, airplane or automobile control, embedded systems, man-machine interfaces, etc. They can be contrasted with interactive systems, which react with the environment at their own speed. This class covers operating systems, data bases, networking, distributed algorithms, etc. Interactive and reactive systems deeply differ on the key issue of behavioral determinism. Interactive systems are naturally viewed as being non-deterministic, while behavioral determinism is a highly desirable and often mandatory of reactive systems. A real-time system is defined as a reactive system that is subject to externally defined timing constraints.

Flight Management Systems (FMS) were first proposed for smart air-crafts in future Air Traffic Management Systems (ATMS)[6] for decentralized air traffic control. FMS are responsible for

- planning of the flight path, generating a proper sequence of flight modes, calculating a feasible trajectory and regulating an Unmanned Aerial Vehicle (UAV) along the nominal trajectory.
- switching among different modes of operation to handle situations like conflict resolution among UAVs, obstacle avoidance, and flight envelope protection.

FMS for UAV[10] are inherently reactive, since they react to the environment by changing mode of operation, as described above.

An FMS operates in a mission critical environment, where reliability and safety are more important criteria than performance. Formal verification and automatic synthesis of implementations are the surest ways to guarantee safety. Managing the design complexity and heterogeneity is the key problem. The design approach should be based on the use of one or more formal models of computation (MOC)[5] to describe the behavior of the system at a high level of abstraction. The implementation of the system should be made using automatic synthesis as much as possible from this high level of abstraction, to ensure implementation that are "correct by construction". Validation should be done at the highest possible levels of abstraction.

The POLIS[2] system is intended for control-dominated systems whose implementation is based on microcontroller for tasks to be implemented in software and Application Specific Integrated Circuits (ASICs) for tasks to be implemented in hardware. The input to POLIS is a combination of graphics and text describing the behavior of each single finite state machine in the formal language ESTEREL. The analysis at the behavioral level can be carried out with formal tools. Performance evaluation can be carried out by simulating the behavior of the architecture selected with an abstract timing model of processor in the heterogeneous simulation environment offered by PTOLEMY[4]. In our design example we carried several simulations with different types of HW-SW partition and choices of microprocessors in order to validate the design.

Section 2 presents formal synthesis of reactive system design. In Section 3, the example design and its representation in the POLIS/Ptolemy domain is discussed in details. Concluding remarks are offered in Section 4.

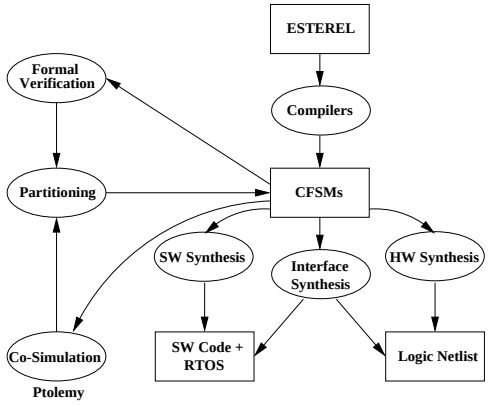

Figure 1: Polis design flow chart

2 Polis Co-design Methodology

In the formal approach to the design of a control system, the choice of the Model of Computation to represent the behavior of the system is often crucial. In order to do it we need to know what are the main characteristics of the system that we are going to implement and then find the MOC that can capture most of the properties [5]. In the case of a reactive system, we need a specification language that includes constructs for hierarchy, preemption (input events arrive at irregular and unpredictable times), concurrency and sequencing. A widely used model for description of such system is the Finite State Machine(FSM).

Definition 1 *A Finite State Machine (FSM) is a quintuple* $\mathcal{F} = (I, O, X, R, F)$ *where* I *is a finite set of input symbols,* O *is a finite set of output symbols,* X *is a finite set of states,* $R \subseteq X$ *is the set of initial states,* $F \subseteq I \times X \times X \times O$ *is the transition relation.*

A FSM contains all the desired properties and also, under mild conditions, it is deterministic and completely specified. The drawback of such a representation is the difficulty posed on the possibility of data computation. The FSM models purely reactive systems, while in almost all control application data computation is also needed.

A more suitable model is the **Extended Finite State Machine** (EFSM), which is a FSM where the transition relation may depend on a set of internal variables. It operates on a set of finite-valued variables by using arithmetic, relational, boolean operators and user defined functions. The EFSM model has a fundamental limitation: communication between EFSMs is totally synchronous, therefore it is not implementable on a distributed environment where a combination of hardware and software modules is used. Hardware and software implementations are characterized by different behavior in execution and communication. The desired MOC should then reflect this situation. The necessity to extend the FSM semantics to include an asynchronous communication mechanism brings to the choice of the so called **Co-Design Finite State Machine** (CFSM). A CFSM can be defined as a FSM which has also a data computation part. CFSM exploits a locally synchronous behavior. It produces an output in reaction to an input assignment in zero time. Globally the CFSM has an asynchronous behavior; each CFSM reads inputs, and produces outputs in an unbounded but finite amount of time. Several specification languages can be used to model CFSMs. POLIS uses synchronous language to model each individual CFSM. The synchronous approach is very attractive for several reasons. Computation and internal communication take no time. The behavior is totally predictable. The problem of synchronization doesn't exist. Determinacy allows formal verification and the synchronous approach allows translation in EFSM in a fully abstract way, so that behaviorally equivalent specifications are mapped into syntactically equivalent EFSMs. Communication between CFSMs happens by means of events which are control signals that may or may not contain also data information.

The high level specification language used by POLIS is **Esterel**[1]. This language is very simple and contains the necessary constructs for the description of our system. Hierarchy is handled via procedure calls, preemption consists of two basic constructs , one which allows the module to terminate its computation for the current instant and one which does not, concurrency is specified by using a parallel composition construct. Data manipulation cannot be done naturally in Esterel. The user needs to define functions outside the environment and then link them in the program. Also the synchronous hypothesis makes difficult to model the communication among subsystems that operate at different rates. The timing constraints need to be specified outside Esterel. This will be assessed during the HW-SW synthesis phase. Starting from the behavioral specification, CFSMs are generated using the Software Hardware Intermediate FormaT (SHIFT) language. The next step is to connect the various CFSMs so generated. This can be done in the simulation environment Ptolemy, which we will discuss later. At this stage formal verification can take place. This technique is very powerful but at the same time computationally expensive. Performing formal verification at the behavioral level allows detection of errors at an early stage and keeps the complexity of the formal verification scheme low.

The next step involves the selection of an implementation architecture. The advantage of using formal methods consists in the use of automatic synthesis techniques. There are three fundamental decisions to be taken: *Partitioning, Architecture Selection and Scheduling*. These three steps are based on experience and therefore the designer is allowed to choose. POLIS provides libraries for several types of micro-controllers for software implementation and ASICS for hardware synthesis. POLIS provides a choice of schedulers, which regulate the communication among CFSMs. After the selection of the architecture is complete the system is simulated within Ptolemy.

2.1 Ptolemy

Co-simulation is used in POLIS both for functional debugging and for performance analysis during the architecture selection process. Hardware-Software (**HW-SW**) co-simulation is generally performed with sepa-

rate simulation models. POLIS allows for HW-SW co-simulation within the same environment. The basic concept is to use synthesized C code to model all the components of a system, regardless of their future implementation. For the software partition the simulation code is the same that will run on the target processor. Depending on the selected architecture, each task will take a specified number of clock to be executed (one clock cycle for hardware, a number of cycles for software depending on the selected target processor for software) and in different execution constraint (concurrency for hardware, mutual exclusion for software). The Ptolemy system provides the simulation engine and the graphical interface. Among the various computation models offered by Ptolemy, the *discrete event* (DE) model has been used, since it matches the CSFM execution semantics. Co-simulation provides a truthful estimate of the performance of the system, i.e. of the capacity of the software architecture to meet the timing constraint imposed by the discretized dynamic system. In case the designer doesn't find the result satisfactory, the aforementioned design process needs to be iterated. If, on the other hand, the simulation results meet the specifications, synthesis can take place. POLIS provides automatic code generation which is specific to the selected micro-processor.

3 Design Example

This section presents an application of this design methodology to the modeling and simulation of a FMS for a UAV performing a particular task. The FMS can be modeled by a hierarchical finite state machine. The mission regards searching for objects of interest in a well-defined area and performing investigation when the object is found. The system can be decomposed into three parts: the Flight Management System which is responsible for planning and controlling the operation of the UAV, a Detector for the detection and investigation of objects of interest and the UAV (the plant to be controlled). Figure 2 shows the planned mission. In order to illustrate the functionality of the system we put an object that the UAV will be able to sense along path.

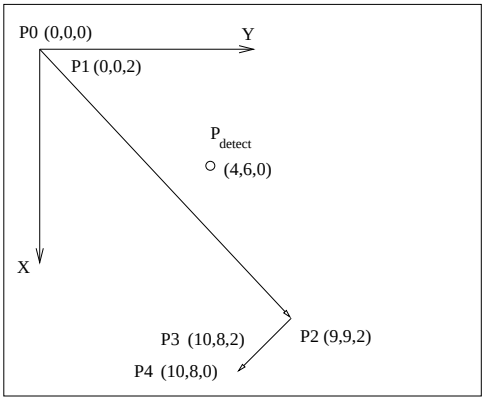

Figure 2: Mission scenario.

The FMS consists of four layers, the strategic, tactical, and trajectory planners, and the regulation layer, as described in Figure 3. Hierarchy is handled very naturally in Esterel, as described in the previous section. Each block is modeled as a FSM by specifying the desired behavior. The individual modules are then composed in Ptolemy to yield to complete system.

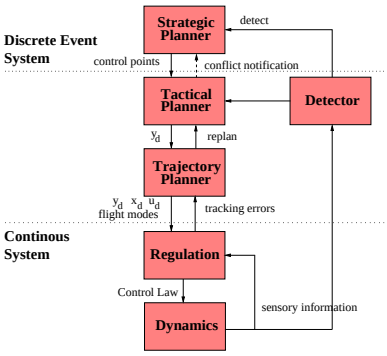

Figure 3: System Architecture

The system architecture in the Ptolemy domain is shown in Figure 4. The stars represent the functional blocks of different modules. In this section we give a description of the role of each of the functional blocks.

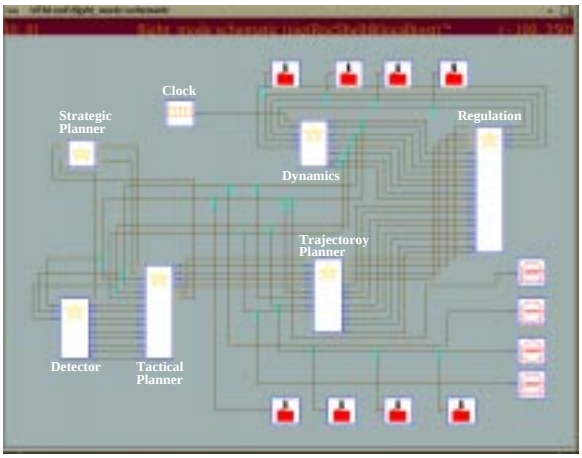

Figure 4: System block diagram in Ptolemy

The **Strategic Planner** is concerned with the planning and execution of the central UAV mission. It designs a coarse, self-optimal trajectory, which is stored in form of a sequence way-points. gives a list of way points that are planned for the mission. This layer also takes care of the transition between the points, by acknowledging the completion of a subtask and scheduling the next one. When an object of interest is detected, the UAV transitions to the investigation operation mode. The strategic planner stores the current way point and resumes when the investigation task terminates. In our example the strategic planner contains a set of way points $P = \{p_0, p_1, p_2, p_3, p_4\}$, where $p_i \in R^3$ for $i = 0..4$.

The **Tactical Planner** is responsible for the coordination and execution of behaviors, and is able to overrule the behavior proposed by the strategic planner, in case of safety critical situations such as collision avoidance, as shown in Figure 7. Given the la-

bels of the set points from the planner it first translates them into actual coordinates using a look-up table, then gives the correct sequence of flight modes needed to achieve the goal. When the detection occurs the tactical planner switches to the investigation mode (Fig. 7) and, given the coordinates of the detected object, introduces a sequence of way points PI_i that will be tracked in order to complete the new task. In our example the UAV will encounter the object while flying from p_1 to p_2, then the new sequence will be $\{p_0, p_1, PI_1, PI_2, PI_3, PI_2, PI_1, p_2, p_3, p_4\}$, where PI_1 is a stop-point, PI_2 is a point above the object and PI_3 is has the same coordinates as the previous point but with lower altitude. Once the investigation has taken place the search continues on the preplanned path.

The **Trajectory Planner** uses a detailed dynamic model, sensory input, and the output trajectory, to design a full state and nominal input trajectory for the UAV, and the sequence of *flight modes* necessary to execute the dynamic plan. The trajectory planner, given the information about the type of flight mode chosen by the tactical planner, executes it choosing the corresponding outputs and the appropriate controllers. Several types of trajectories can interpolate the way points. In our design only a piece-wise continuous trajectory is implemented, where the UAV stops at each point, as shown in Figure 5.

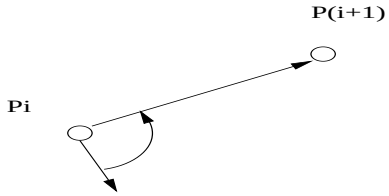

Figure 5: Piecewise linear trajectory

This strategy is the simplest and safest one because it completely decouples the manœuvres and performs them independently. Future work will include a higher variety of different trajectories. The choice between them can then be done on-line according to objectives such as avoiding extremely aggressive manœuvres (to maintain the uncontrollable modes inside the range of stability), collision avoidance, minimum fuel consumption, minimum travel time and so on. At the completion of the submission the Tactical Planner sends the acknowledgment signal to the Strategic Planner which sends the next way point. The transition is restricted in proper sequence which is shown in Figure 6.

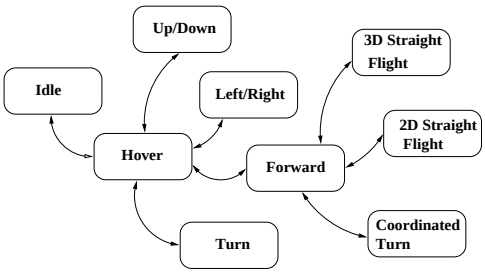

Figure 6: Transitions diagram

Each basic flight mode has then associated with it a particular combination of control actions. Each time an elementary flight mode becomes active the corresponding type of controls are requested to be used at the regulation layer.

The **Regulation Layer**, together with the plant, represents the continuous part of the system. The regulation layer has access to sensory information about the actual state of the UAV to generate control signal to regulate it on the given trajectory. It consists of the so-called auto-pilot, which receives the flight mode and computes the input signals to the vehicle according to the system state. There are four inputs to the plant, so only four of the six degrees of freedom can be controlled. Since we are assuming that input-output linearization is applied in the inner loop, the control signals are used to control the four selected outputs directly. After applying input-output linearization, the resulting system is linear and a linear control law is applied.

The **Detector** has limited range of detection capability. The image processing computing time together with the view angle will give an upper bound on the maximum cruise velocity. The detector communicates with the tactical planner and the strategic planner. When detection takes place, a preemptive signal is sent to the tactical planner which switches to the investigation operational mode, as shown in Figure 6. Once the investigation is accomplished, the detector will assert a signal to the tactical planner to indicate the end of investigation phase.

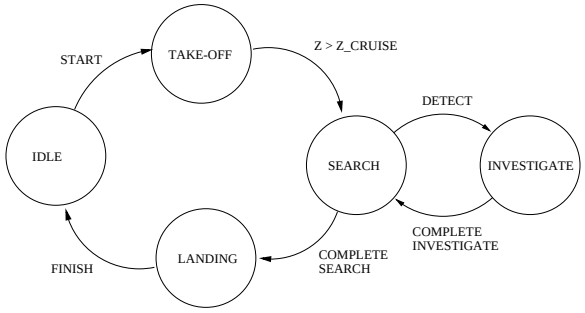

Figure 7: FSM representation of Tactical Planner

An aerial vehicle is represented by a rigid body moving in a 3–dimensional space in response to gravity, aerodynamics, and propulsion. Due to the characteristics of the aerial vehicle designed, the force and moment are generated by different actuators which corresponding to different control inputs. For aircraft, the control inputs are generated through engine, aileron, elevator and rudder which produce thrust and 3-axis moments. For an helicopter, the control inputs are generated through engine, main rotor collective pitch, tail rotor collective pitch, longitudinal cyclic pitch, lateral cyclic pitch, which produce thrust and 3-axis moments.

Following the work done by [8], the input–output system is decoupled and linear. For our modeling purpose, we could then use a second order linear system for each output variable to model the **Dynamics** as:

$$\begin{aligned} \ddot{x} &= u_x \\ \ddot{y} &= u_y \end{aligned}$$

$$\ddot{z} = u_z$$
$$\ddot{\psi} = u_\psi .$$

This block then receives the control signals as inputs and computes the evolution of the states using the Newton Forward method of integration. Differential flatness has been applied on approximate models of aircraft[3, 11] and helicopter[8] in generating trajectory. The flat outputs, positions and heading, have been used to feedback-linearize the approximated helicopter model. Thus, the feedback-linearized system can be treated as chains of integrators with redefined inputs. By making use of the differential flatness property, full states and nominal inputs can be recovered from the outputs and their derivatives. Hence, any violation in the state and input constraint can be detected and a new trajectory can be generated. In our case pitch and roll trajectories can be reconstructed from outputs and inputs.

4 Simulation and Implementation

To validate the design, we carried out several simulations under different assumptions. The behavior simulated was a search-and-investigate mission. The UAV should deviate from the nominal path if the detector finds an object of interest. The initial states are set to zero, i.e. idle position. The mission terminates when the UAV lands at the point p_4 (10,8,0).

First, as depicted in Figure 8, we simulated the behavior of the system under the synchronous hypothesis, i.e. in the absence of delay due to communication and computation time. This corresponds to representing the system entirely in Esterel. This assumption is often used to simulate the *behavior* of the system but yields poor implementations since it requires implementing the system either in synchronous hardware or as a single task in software. This results in overheads in space for hardware and in memory and sometimes in running time for software.

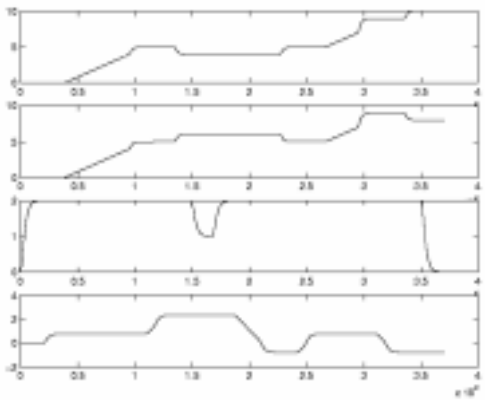

Figure 8: Simulation result: Synchronous model: x, y, z, ψ

After the behavior has been simulated and verified, a specific implementation has to be selected. Our first architectural choice was to implement the algorithm entirely in software on a RISC MIPS R3000 microcontroller with a simple round-robin scheme to schedule the tasks. The synchronous assumption was maintained for the plant and the detector, thus implying that their operation was considered as a single task for the operating system. The simulation results in Figure 9 show that the timing constraint is met by the chosen architecture. Simulation of 8 minutes of actual running time of the system was obtained with 5 minutes of simulation time. At first sight, this result seems surprising since simulation required less time than the actual time that would have been required by the embedded system. The explanation of this result is simple: the software is run on a workstation whose processor is more powerful than the micro-controller core!

The timing results needed to verify the correctness of the implementation with respect to the real time constraints are obtained using the performance model used for the controller. In particular, the number of cycles required by each instruction in the instruction set of the processor have to be provided. Because of the complex architecture of modern microprocessors, the POLIS group has developed a method to obtain the parameters of the performance model by running on an evaluation board a set of tests.

The actual running time estimates provided by the simulation are not 100% accurate but it has been possible to demonstrate that the accuracy of the estimation is acceptable (the actual running time on the implemented system was within 20% of the estimation).

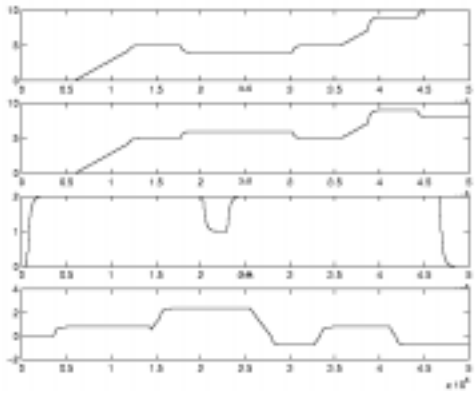

Figure 9: Simulation Result after HW-SW partition: x, y, z, ψ

Because of the speed of the simulation with the performance estimation models (which include a model for the operating system and the scheduler as well), several different implementation choices can be tried. In particular, different micro-controllers and different scheduling algorithms can be evaluated. Once several architectures have been validated, the designer can choose the most suitable architecture, depending on cost efficiency trade off consideration.

To exemplify this point, we implemented the system using a Motorola 68hc11, 8 bit micro-controller and a round robin scheduler. The implementation yields an unstable system, as shown in figure 10. Analysis on the dynamical system shows that bandwidth of the system is too high to achieve acceptable performance with a slow processor. In particular, part of the design has to

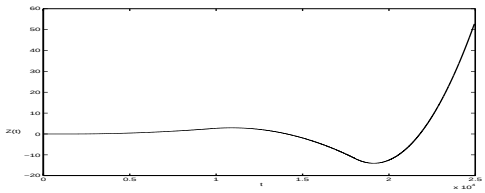

Figure 10: Unstable Z dynamics

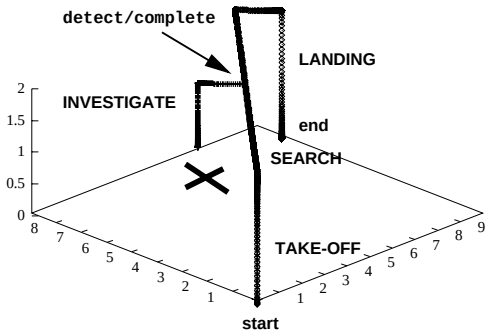

Figure 11: Simulated 3D Trajectory

be implemented in hardware to meet the design specification with such architecture. Further simulations show that the design specification is met if the regulation layer is implemented with an ASIC. The switching condition between flight modes needs to be relaxed, since the state information is not available to the tactical planner at all times.

5 Conclusion

A formal approach to reactive system synthesis using POLIS has been presented. A design example of FMS of UAV has been exploited. For such a hierarchical system with complex behavior, the behavior is specified in Esterel and compiled into a network of CFSMs chosen as the MOC for describing a locally synchronous and globally asynchronous system. Formal verification can be carried out with the CFSM model. After verification of the desired design properties, partition of hardware and software can take place under the choice provided by the designer. Automatic synthesis of hardware, software and interface, including the Real Time Operating System (RTOS), is then performed. In this experiment, we found that a great value of the POLIS system is in the quick system-level analysis that allows an architectural optimization that would not be possible otherwise especially for complex systems such as the ones analyzed in this paper.

The design is validated through simulation in Ptolemy environment. However, it is clear that final validation has to be carried out in a prototype implementation of the entire system since the high-level simulation carried out in the Ptolemy environment with POLIS models and software synthesis methods is based on approximate performance models. It is our intention to build such a system with the micro-processor selected in the evaluation phase with the scheduling algorithm tested in the simulation.

Acknowledgment

Research is supported by grants (ARO)DAAH04-96-1-0341, (DARPA)F33615-98-C-3614, (ONR)N00014-97-1-0946 and CNR. The authors would like to acknowledge Marco Sgroi for his invaluable advice.

References

[1] G. Berry. The foundations of esterel. In C. Stirling G. Plotkin and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1998.

[2] F.Balarin, P.Giusto, A.Jurecska, C.Passerone, E.Sentovich, B.Tabbara, M.Chiodo, H.Hsieh, L.Lavagno, A.Sangiovanni-Vincentelli, K.Suzuk. *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, 1997.

[3] M. Fliess. Aircraft control using flatness. In *Proceedings of Symposium on Control, Optimization and Supervision*, volume 1.2 vol. 1322, Lille, France, July 1996.

[4] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. Journal of Computer Simulation, special issue on Simulation Software Development*, 4:155–182, April, 1994.

[5] E. A. Lee and A.Sangiovanni-Vincentelli. Comparing models of computation. In *ICCAD*, 1996.

[6] S. Sastry, G. Meyer, C. Tomlin, J. Lygeros, D. Godbole, and G. Pappas. Hybrid control in air traffic management systems. In *Proceedings of the 1995 IEEE Conference in Decision and Control*, pages 1478–1483, New Orleans, LA, December 1995.

[7] S.Edwards,L.Lavagno,E.A.Lee and A.Sangiovanni-Vincentelli. Design of embedded systems: Formal models, validation, and synthesis. *Proceedings of the IEEE*, 85, No. 3, March, 1997.

[8] T. J. Koo, and S. Sastry. Output tracking control design of a helicopter model based on approximate linearization. In *Proceedings of the 37th IEEE Conference on Decision and Control*, Tampa, Florida, December 1998.

[9] T. J. Koo, D. H. Shim, O. Shakernia, B. Sinopoli, Y. Ma, F. Hoffmann, S. Sastry. Hierarchical hybrid system design on berkeley uav. In *International Aerial Robotics Competition*, Richland, Washington, USA, August 1998.

[10] T. J. Koo, F. Hoffmann, H. Shim, B. Sinopoli, and S. S. Sastry. Hybrid control of model helicopter. In *Proceedings of IFAC Workshop on Motion Control*, Grenoble, France, October 1998.

[11] M. J. van Nieuwstadt and R. M. Murray. Fast mode switching for a thrust vectored aircraft. In *Proceedings of Multiconference on Computational Engineering in Systems Applications*, Lille, France, July 1995.