Open access • Proceedings Article • DOI:10.1109/OCEANS.2007.4449142

# Automated Coordinator Synthesis for Mission Control of Autonomous Underwater Vehicles — Source link

Siddhartha Bhattacharyya, Ratnesh Kumar, S. Tangirala, Lawrence E. Holloway

**Institutions:** Kentucky State University, Iowa State University, Pennsylvania State University, University of Kentucky

**Published on:** 01 Sep 2007 - OCEANS Conference

**Topics:** Intervention AUV, Mission control center and Remotely operated underwater vehicle

# Automated Coordinator Synthesis for Mission Control of Autonomous Underwater Vehicles

S. Bhattacharyya[+], R. Kumar[*], S. Tangirala[#], and L. E. Holloway[x]
+ Kentucky State University (email:s.bhattacharyya@kysu.edu)
* Iowa State University (email:rkumar@iastate.edu)
# Applied research Laboratory, Pennsylvania State University (shaky@psu.edu)
x University of Kentucky (holloway@engr.uky.edu)

*Abstract*— **In our past work we have developed a hierarchical hybrid-model based mission control approach for autonomous underwater vehicles. The approach is aided by tools that allow graphical design, iterative redesign, and code generation for rapid deployment onto the target platform. The goal is to support current and future autonomous underwater vehicle (AUV) programs to meet evolving requirements and capabilities. The hierarchical architecture contains mission controllers at each level which coordinate with other controllers, the vehicle, and the user for the successful execution of a mission. Here we propose an approach for automated synthesis of such controllers, and illustrate by applying the algorithm for automated synthesis of the highest-level coordinators.**

## I. INTRODUCTION

Many practical systems can be modeled as a group of interacting hybrid systems. A growing need for modeling, design and analysis of such systems has led to an increased interest for research in this area. In our past work we have developed a hybrid-model based hierarchical mission control architecture for autonomous underwater vehicles (AUVs) that facilitates graphical design and code generation [10], verification of logical correctness [11], and animation of the AUV depicting the missions executed [18].

The control tasks for an autonomous underwater vehicle is divided into lower level control, concerned with control of continuous vehicle dynamics and a higher-level mission control, which has discrete real-time dynamics and is concerned with safe execution of mission. The overall control is a hybrid system containing both continuous and discrete dynamics.

The basic idea is to hierarchically decompose missions into sequence of operations, and operations into sequence of behaviors, and behaviors into sequence of vehicle maneuvers. As shown in Figure 1 at the lowest level of the hierarchy is the underwater vehicle (plant) along with the vehicle controllers (VCs) above which we have the mission controller. The lowest level of the mission controller is comprised of *Behavior Controllers*, where a *behavior* may be thought of as a skill or ability that an autonomous system possesses which enables it to perform specific mission tasks (*thrive*) while remaining safe (*survive*). Behaviors require execution of sequences of vehicle maneuvers. The middle level of the mission control hierarchy consists of *Operation Controllers*, where an *operation* represents a mission segment or phase that is integral to the completion of the overall AUV mission. Operations, command/sequence the behavior controllers to achieve their objectives. The highest level of the mission controller consists of the *Mission Coordinators* which are responsible for sequencing and scheduling operations in order to complete the mission while ensuring the safety of the vehicle. Controllers at each of the levels coordinate those at the lower levels to achieve a higher level behavior or operation or mission, as the case may be. Modules within a level may communicate with each other and each level in the hierarchy is restricted to command the level immediately below it and send responses to the level immediately above it. All levels in the mission controller hierarchy may assign vehicle commands directly by placing an appropriate vehicle command in the shared database.

Hierarchical approach reduces the complexity of design and also facilitates the verification, animation and automated synthesis of the highest level mission controller module(s), which is the main theme of this paper. Here we present a method for the automated synthesis of the coordinators.

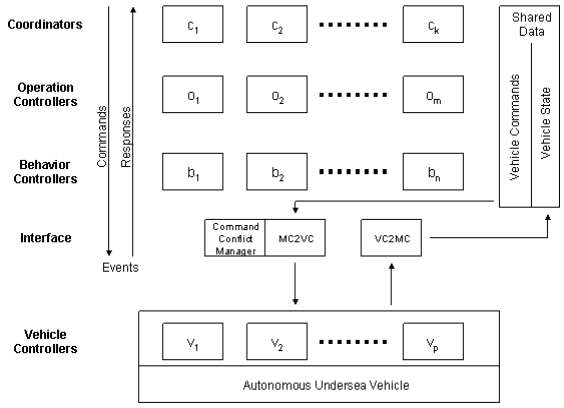


Figure 1: Hybrid Mission Control Architecture

The vehicle controller and the mission controller communicate through an interface layer symbolically represented by MC2VC (mission controller to vehicle controller) and VC2MC (vehicle controller to mission controller). The MC2VC block also includes a Command Conflict Manager which is responsible for selecting a specific

vehicle level command (when more than one exists) according to a static or dynamic priority list or using other methods (such as optimization). This module is included since all modules in the mission controller hierarchy are allowed to assign vehicle commands directly, and so there is a distinct possibility that multiple vehicle commands can coexist.

A sequence of commands is sent to the vehicle subsystem controllers via the MC2VC interface. AUV state information is collected by the sensors and transferred by the VC2MC interface periodically to the shared database. This state information is made available to all modules in all levels of the mission controller hierarchy. Similarly, vehicle commands, assigned and manipulated by all levels in the mission controller are stored in the shared database and sent to the AUV by the MC2VC interface.

An event is initiated by a particular module and its recipients are controlled by an event dependency table which may be static or dynamic. The entire mission controller contains interacting hybrid automata, which is formally defined in a section below.

The mission controller modules are developed using TEJA software tool [5], which supports the design of interacting hybrid state machines and includes automatic real-time code generation allowing for a rapid deployment on the target platform. For verification purposes, the Teja modules specifications are first transformed [6] into a format readable by Uppaal [6] and HyTech [8], a hybrid system modeling, simulation, and verification tool. For animation, the mission controller modules in Uppaal are further converted to animation modules of OpenGL [9].

In the present work our goal is to propose the automated synthesis of mission coordinators (i.e. controllers at the topmost layer) for hierarchy based hybrid mission control architecture for AUVs. The interactions within the modules in the hierarchical control architecture are complex. Synthesis of a coordinator for such a system is a challenging task as it requires careful monitoring of the inputs received and the outputs sent. The controller is a hybrid system with discrete states and continuous dynamics. The continuous dynamics are implemented as functions. The coordinators are a special case of hybrid system with timing constraints and are known as timed automata.

Our method of coordinator synthesis is based on identifying constant and variable properties like application specific events. The basic idea for synthesis of coordinators is such that it should satisfy properties to execute missions.

In section II we discuss the hierarchical mission controller architecture for the survey AUV implemented at the Applied Research Lab at Pennsylvania State University. In section III the hybrid system model is discussed and in section IV we discuss our approach of coordinator synthesis, in section V we discuss related work and all that has been accomplished in our work and finally conclude our work in section VI.

## II. MOTIVATING APPLICATION: A SURVEY AUV

Figure 2 shows the details of a specific application of the general AUV mission control architecture to a generic *survey AUV*. The primary mission of a survey AUV is to transit to a user specified location and conduct a survey following a specific pattern in 3D, at a specified speed and depth/altitude. In this example, there are three vehicle controllers (VCs), the *Autopilot* which accepts commands to control the altitude, speed and depth of the AUV; the *Variable Buoyancy System (VBS) Controller* which accepts commands to control the trim and buoyancy of the AUV; and the *Device Controller* which accepts commands to control the various sensors and other devices on board the AUV.

The lowest level of this mission controller is comprised of two behavior controllers: *Steering*, which is responsible for steering the vehicle to a specified location in space and interacts with the Autopilot; *Loiter* which controls the vehicle to loiter at a specific location in space for a specified duration and interacts with the Autopilot and VBS Controller.

The behavior controllers are, in turn, commanded by the operation controllers, which correspond directly to mission orders that are specified by the user and are described next. The *Pause* operation controller is used under certain situations to let the vehicle remain at it's current state for a specified duration. The *Launch* operation controller is responsible for bringing the vehicle off of the surface and running at depth with enough forward speed to achieve controllability. This controller interacts with the Autopilot, the VBS Controller, and the Device Commander controller. The *GPSFix* operational controller sequentially commands the AUV to shut off propulsion, rise to the surface, raise the GPS mast, obtain a GPS-aided position fix retract the GPS mast, and re-launch the AUV. This controller interacts with the Autopilot, behavior controller, the Device Commander, the Device Controllers, and the Launch operation controller. The *WaypointNavigator* operation controller controls the AUV to transit to waypoints specified by the mission specification. This controller interacts with Steering, Loiter, and the Device Controller. The *Device Commander* is used to control sensors and devices on the AUV in response to mission orders; this controller interacts with the Device Controllers. Finally, at the highest level of the AUV mission controller are the mission coordinators of which there are two types: *Progress* and *Safety*, where the progress coordinator is divide into two parts: *Sequential*, and *Timed*. The sequential coordinator is responsible for executing a mission consisting of a sequence of operations; a timed coordinator is responsible for executing a timed sequence of operations; and a safety coordinator ensures safe operation.

According to our hierarchical architecture coordinator synthesis involves synthesizing the top level controllers which receive input from the lower level controllers and the mission file. The functionality of all the coordinators together is to send command to the lower level controllers, respond to successful mission completion and react to exceptions like aborting missions or modifying parameters for safety.

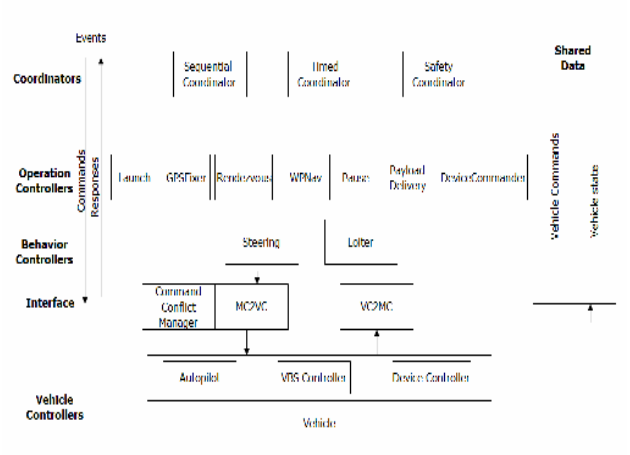


Figure 2: Survey mission control architecture

## III. HYBRID SYSTEM MODEL: NOTATIONS AND PRELIMINARIES

Hybrid systems are systems, which include continuous as well as discrete signals and states. Hybrid systems [4] [10] have been used as mathematical models for many important applications. Their wide applicability has inspired a great deal of research from both control theory and theoretical computer science [7].

An AUV is a hybrid dynamical system with both discrete and continuous states. Hybrid systems can be modeled as hybrid automata. A hybrid automaton model captures the evolution of variables over time. The variables will evolve continuously as well as in instantaneous jumps. A hybrid automaton is as described below. This type of modeling formalism has been used to model the underwater vehicle control modules.

### A. *Controlled hybrid automaton*

A *controlled* hybrid automaton is a tuple $\mathcal{H}=(Q,\Sigma,U,Y,F,H,I,E,G,R)$ consisting of the following components:

**State space:** $Q=L\times X$ is the state space of the hybrid automaton, where $L$ is a finite set of locations and $X=\Re^n$ is the continuous state space. Each state $Q$ can be described by $(l,x)\in Q$, where $l\in L$ and $x\in\Re^n$.

**Events:** $\Sigma$ is the finite alphabet or event set of $\mathcal{H}$.

**Continuous Controls and Parameters**: $U=\Re^m$ is the continuous control space consisting of control signals and exogenous continuous-time parameters. $u:[0,\infty)\to U$ denotes a control vector comprised of these parameters.

**Outputs:** $Y$ is the output space of $\mathcal{H}$, which may consist of both continuous and discrete elements.

**Continuous Dynamics:** $F$ is a function on $L\times U$ assigning a vector field or differential inclusion to each location and continuous control vector. We use the notation $F(l,u)=f_l(u)$.

**Output Functions:** $H$ is a set of output functions, one for each location $l\in L$. We use the notation $H(l)=h_l$, where $h_l:X\times U\to Y$ is the output function associated with location $l\in L$.

**Invariant conditions:** $I\subset 2^X$ is a set of invariant conditions on the continuous states, one for each location $l\in L$. We use the notation $I(l)=i_l\subseteq X$. If no $i_l$ is specified for some $l\in L$, then it's default value is taken to be $X=\Re^n$.

**Edges:** $E\subset L\times\Sigma\times L$ is a set of directed edges. $e=(l,\sigma,l')\in E$ is a directed edge between a source location $l\in L$ and a target location $l'\in L$ with event label $\sigma\in\Sigma$. In addition, $E=E_c\cup E_\Phi$, where $E_c$ and $E_\Phi$ represent the controlled and uncontrolled edges, respectively.

**Guard conditions:** $G\subset 2^X$ is the set of guard conditions on the continuous states, one for each edge $e\in E$. We use the notation $G_e=g_e\subseteq X$. If no $g_e$ is explicitly specified for some edge $e\in E$, then it's default value is taken to be $X=\Re^n$.

**Reset conditions:** $R$ is the set of reset conditions, one for each edge $e\in E$. We use the notation $R(e)=r_e$, where $r_e:X\to 2^X$ is a set-valued map. If no $r_e$ is explicitly specified for some edge $e\in E$, then the default value is taken to be the identity function.

The semantics of a hybrid automaton can be understood as follows. When in a certain discrete configuration *l,* the continuous-state *x* of the hybrid system evolves according to the controlled vector-field $F_l(x,.)$. The evolution of the continuous-state according to the flow of $F_l(x,\mu.)$ is defined as long as *x* lies in the domain specified by the invariant condition $i_l$. If at anytime during its evolution the continuous-state acquires a value that satisfies a guard condition $g_e$ for some edge $e=(l,\sigma,l')$ of the hybrid automaton, the system can transition from configuration *l* to *l'*. The transition is labeled by an ``event" $\sigma$ and the continuous-state in the new configuration acquires a value specified by the reset condition $r_e$. When in new configuration *l'* the continuous-state evolves according to the controlled vector-field $F_{l'}(x,.)$. In the next we discuss the algorithm to synthesize such hybrid coordinators.

## IV. PROPOSED APPROACH FOR COORDINATOR SYNTHESIS

Automated synthesis of the mission controllers promises reduction in time to develop and implement controllers for hierarchical control of autonomous vehicles. It also improves modification and debugging capability.

Our goal in the automation of controllers is to translate the mission specifications and user inputs into sequence of actions to successfully execute the mission. The specifications are the sequence of operations for the coordinators, the sequence of behaviors for the operation controllers and the sequence of vehicle maneuvers for behavior controllers. The coordinators we synthesize are timed automata with timing constraints modeled as guard conditions. The operation and behavior controllers are hybrid systems implementing discrete states

with continuous dynamics. The definitions of the automata built are as discussed in section III.

The controllers consist of a basic structure which is common to all the controllers (excepting the safety coordinator which involves a basic structure) and a specific synthesized structure based on the specific operation, behavior or maneuver to be executed. Examples of the basic structure are responding to failures or harmful events or initializing when started. An example of a specific mission can be to find the present location using a GPS or fire a missile. Although these two distinct structures are exhibited by the controllers at each level we only discuss the algorithm for synthesizing the coordinators. We consider the requirement of three coordinators at the topmost level. The three coordinators are a sequential coordinator (implementing sequential control to execute a sequence of actions for a mission), a timed coordinator (implementing time critical missions) and a safety coordinator (implements safe execution of mission). These coordinators are synthesized based on user input and high level specification.

*A. Sequential coordinator*

Sequential coordinator coordinates the execution of sequential untimed mission. The synthesis of sequential coordinator is based on the inputs received and the way it responds to mission execution. Inputs received by the sequential coordinator can be requests made by the user i.e. the mission order or other coordinators at the same level or responses received from lower level or same level coordinators. The algorithm consists of two parts the first part implements the basic structure and the second part implements the augmentation of new edges, guards, reset values and locations to the sequential coordinator for the mission specific structure. The simplest structure of the sequential coordinator is shown in Figure 3 showing the command (Do) and responses (Done) and the user input. The basic structure is the same for all the coordinators which involves two different phases: Initialization phase, and Communication establishment phase. The mission specific structure contains mission order phase, and a response phase.

Initialization phase involves synthesizing the states (invariants) and transitions (guards, events, resets) of the coordinators that should occur during system start up. During the communication phase communication is established with the remote station to receive mission order. To handle the mission specific structure the mission order phase models the states and transitions of the sequential coordinator based on the operations. The response phase involves modeling the way the sequential coordinator responds to successful completion of operations or handling exceptions like failure or termination of operations/missions. In the algorithm explained next the label Start indicates the beginning of the mission specific implementation.

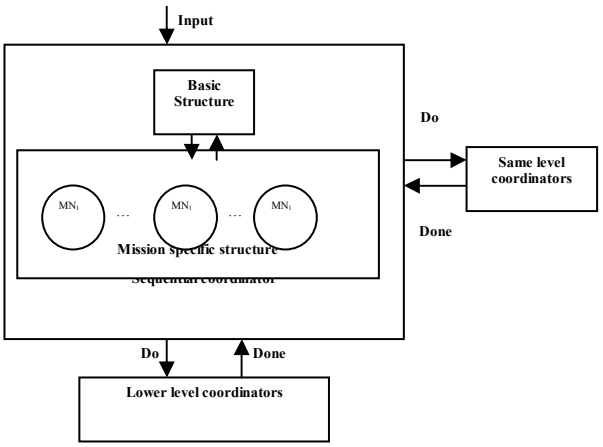


Figure 3: Structure of Sequential Coordinator

Algorithm:

Create five locations $l \in L$ and name them as ***Idle, WaitforVCComms, Run, Suspend*** and ***Endmission***. (control for any AUV needs all these states )

- Create an edge $e_0$ from ***Idle*** to ***WaitforVCComms*** (indicating transition to a state to wait to establish communication with the Vehicle)
  - Set event $\sigma_{in} = Init$
  - Set guard condition $G^i(e_0) = t>=T$ where $T$ is a constant time to initialize the system (for our case $T = 1$)
  - Set reset condition $R(e_0) = \{t=0\}$
- Create an edge $e_1$ to ***Run*** state from ***WaitforVCComms*** if a connection with vehicle is established
  - Set event $\sigma_{in} = NewVCData$
  - Set guard condition $G^i(e_1) = t>=10$
  - Set reset condition $R(e_1)$ = {t = 0, MissionTime = 0, Suspendable = 0}
- Create an edge $e_2$ from ***Run*** state to ***EndMission*** state
  - Set event $\sigma_i = Endmission$
  - For each $Controller_i \in Level_j$ where i= 1…n, j = 1 only
    - Set the guard condition on the edge $G^i(e_2)$ = ( $\cup$ $Controller_k$->Idle) where k = 1,2…n, k != i checking the status of other controllers (0 meaning idle)
  - Set reset condition $R(e_2)$ = {t = 0, Suspendable = 0, Idle = 0} to indicate that all the coordinators are idle
- Create an edge $e_3$ from ***Run*** state to ***EndMission*** state
  - Set event $\sigma_i$ / $\sigma_o$ = *Abort / Abort*
    - Set guard condition $G(e_i) = \{True\}$
    - Set reset condition $R(e_i) = \{t = 0\}$
- At ***EndMission*** state
  - Draw a self loop edge $e_4$
    - Set event $\sigma_{in} = OnSurface$
    - Set guard condition $G^i(e_3) = (Var_k <= SurfaceThreshold)$ where $Var_k$ is $k^{th}$ variable mapped to set of real

numbers (indicating sensor value of depth) *SurfaceThreshold* indicates a constant value

- **Begin:** Check if mission order file has more operation names (based on the format used for separating operation names commas mean there are further operations or blank means no operation).
- If no operations in the order file go to **End**
- **Start:** Get Mission order name **$Or_n$(<OperationName>, Prm)**.
- If mission name is obtained for the first time
  - Create a location $l \in L$ and name it **<OperationName>**
  - Draw an edge $e_i$ from the ***Run*** state to **<OperationName>** state where i = j+1… n, where j is the number for the last edge that was drawn
    - Set the events as $\sigma_{in}/\sigma_o$ = **<OperationName> /Do<OperationName>** command sent to the lower order controllers

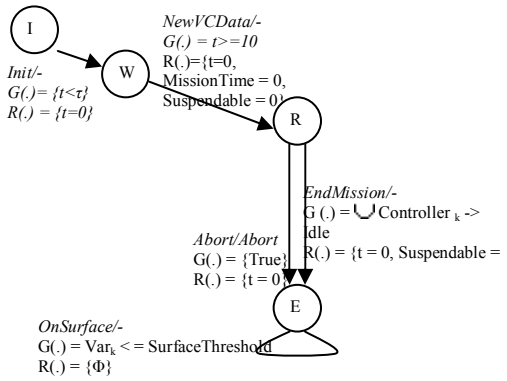


Figure 4: Basic Structure of sequential coordinator

  - Set the guard condition $G^j(e_i)$ = $\{Var_k$ = **"<OperationName>"}**
  - Set the reset condition R($e_i$) ={Suspendable = (0 or 1), Idle = 0, t = 0}
  - If **Suspendable** = 1
    - Create an edge $e_i$ from **<Operation Name>** state to ***Suspend*** state
      - Set event $\sigma_{in}$ = *Suspend*
      - Set guard condition $G(e_i)$ = *{True}*
      - Set reset condition $R(e_i)$ = *{t = 0, Suspendable = 0}*
    - If connecting to the ***Suspend*** state for the first time
      - Create a self loop $e_i$ at the ***Suspend*** state
        - Set the event $\sigma_{in}$ / $\sigma_o$ = *Abort / Abort*
        - Set guard condition $G(e_i)$ = *{$Var_k$ = !Suspended }*
        - Set reset condition $R(e_i)$ = *{Suspended = 1}*
    - Create an edge $e_i$ from ***Suspend*** state to ***Run*** state
      - Set event $\sigma_{in}$ = *Resume*
      - Set guard condition $G(e_i)$ = *{True}*
      - Set reset condition $R(e_i)$ = *{Suspended = 0, Suspendable = 0, t = 0}*
  - Draw an edge $e_i$ from the **<Operation Name>** state to ***EndMission*** State
    - Set the $\sigma_{in}/\sigma_o$ = *Abort / Abort*
    - Set guard condition $G(e_i)$ = *{True}*
    - Set reset condition $R(e_i)$ = *{t = 0}*
  - Create an edge $e_i$ from **<OperationName>** state to ***Run*** State
    - Set event $\sigma_{in}$ = **<OperationName>Done**
    - Set guard condition $G(e_i)$ = *{True}*
    - Set reset condition $R(e_i)$ = *{t = 0, Suspendable = 0}*
- Else if operation name is already there
  - Go to **Begin** to get the name of the next mission
- **End**

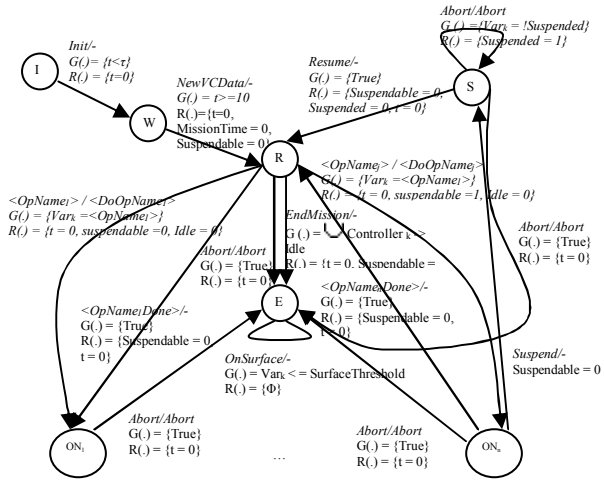


Figure 5: Sequential Coordinator

## B. *Timed coordinator synthesis*

Timed coordinator coordinates the execution of time critical mission. Time critical mission involves execution of sequence of operations with timing constraints. The timed coordinator is synthesized based on the inputs received and the way it should

react to responses it receives. Based on the operation to be executed one other action the timed coordinator needs to implement is to check whether the sequential coordinator should be suspended or not and then performing the action as required. Inputs received by the timed coordinator can be requests made by the user i.e. the mission order or other coordinators at the same level or responses received from lower level or same level coordinators. Response is the way the timed coordinator reacts to a specific situation (similar to the way explained for sequential coordinator). The algorithm till the label Start implements the basic structure and the remaining part implements the augmentation of new edges, guards, reset values and locations to the sequential coordinator. Algorithm:

- Create seven locations $l \in L$ and name them as ***Idle, WaitForFirstTO, CheckOrders, Wait4Suspend, Check4Resume, Decide*** and ***End***.
- Draw an edge $e_0$ from ***Idle*** state to ***WaitForFirstTO*** state (wait for timed order to arrive)
  - Set event $\sigma_{in}$ = *Init/-*
  - Set guard condition $G(e_0) = \{t >= \tau\}$ where $\tau$ is a constant
  - Set reset condition $R(e_0) = \{ \Phi\}$
- Draw an edge $e_1$ from ***WaitForFirstTO*** to ***CheckOrders*** (check for timed order)
  - Set event $\sigma_{in}$ = *NewVCData/-*
  - Set guard condition $G(e_1) = \{True\}$
  - Set reset condition $R(e_1) = \{MissionTime = 0, t = 0, Done = 1\}$
- Draw an edge $e_2$ from ***CheckOrders*** state to ***End*** state
  - Set event $\sigma_{in}$ = *EndMission*
  - Set guard condition $G(e_2) = \{True\}$
  - Set reset condition $R(e_2) = \{Idle = 0\}$
- Draw an edge $e_3$ from ***CheckOrders*** state to ***Decide*** state (to check the requirement to suspend the Sequential cordinator)
  - Set event $\sigma_{in}$ = *NewOrder*
  - Set guard condition $G^i(e)$ = strcmp(this->CurrTimedOrd->Name,"None") && TimedActions_get_MissionTime() >= this->CurrTimedOrd->Time && (!TimedActions_CheckSuspend(this)||this->SeqController->Idle||this->SeqController->Suspended)
  - Set reset condition $R(e_3) = \{Idle = 0\}$
- Draw an edge $e_4$ from ***CheckOrders*** to ***Wait4Suspend*** (indicating that the mission requires suspension of the other coordinators)
  - Set $\sigma_{in}$ = *Suspend/ Suspend*
  - Set guard condition $G(e_4) = \{$ *strcmp(this->CurrTimedOrd->Name,"None") && TimedActions_get_MissionTime() >= this->CurrTimedOrd->Time && (TimedActions_CheckSuspend(this)&&this->SeqController->Suspendable && !this->SeqController->Idle) &&!this->SeqController->Suspended}*
  - Set reset condition $R(e_4) = \{t = 0, Idle = 0, Time2Suspend = 0\}$
- Create a loop $e_5$ at ***Wait4Suspend*** state
  - Set $\sigma_{in}$ = *Suspend/ Suspend* (suspend the Sequential Coordinator)
  - Set guard condition $G^i(e_5)$ = !this->SeqController->Suspended
  - Set reset condition $R(e_5) = \{t = 0\}$
- Draw an edge $e_6$ from ***Wait4Suspend*** to ***Decide*** state
  - Set $\sigma_{in}$ = *NewOrder*
  - Set $G^i(e_6)$ = this->SeqController->Suspended
  - Set reset condition $R(e_6) = \{\Phi\}$
- Draw an edge $e_7$ from ***Check4Resume*** to ***CheckOrders*** without *Resume* event
  - Set $\sigma_{in}$ = *OrderComplete/-*
  - Set the $G^i(e_7)$ = !this->SeqController->Suspended || (TimedActions_get_MissionTime() >= this->CurrTimedOrd->Time && strcmp(this->CurrTimedOrd->Name,"None"))
  - Set the reset condition $R(e_7) = \{\Phi\}$
- Draw an edge $e_8$ from ***Check4Resume*** to ***CheckOrders***
  - Set $\sigma_{in}$ = *OrderComplete /Resume*
  - Set $G^i(e_8)$ = this->SeqController->Suspended && (TimedActions_get_MissionTime() < this->CurrTimedOrd->Time || !strcmp(this->CurrTimedOrd->Name,"None"))
  - Set the reset condition $R(e_8) = \{\Phi\}$
- Draw an edge $e_9$ from each of the states (excepting ***Idle*** and ***WaitForFirstTO***) to ***End*** state
  - Set event $\sigma_{in}$ / $\sigma_o$ = *Abort/Abort*
  - Set guard condition $G^i(e_9) = \{True\}$
  - Set reset condition $R(e_9) = \{\Phi\}$
- **Begin:** Check if mission order file has more operation names
- If no operations in the order file go to **End**
- **Start:** Get Operation names from Mission order file **$Or_n$(<OperationName>, Prm)**.
- If mission name is obtained for the first time
  - Create a location $l \in L$ and name it **<OperationName>**
  - Draw an edge $e_i$ from the ***Decide*** state to **<OperationName>** state where i = j+1…n where j is the number for the last edge drawn
    - Set $\sigma_{in}$ / $\sigma_o$ = **<OperationName>/ Do<OperationName>** sent to lower level controllers
    - Set guard condition $G(e_i)$ = *{CurrentOrder = <OperationName>}*
    - Set reset condition $R(e_i) = \{\Phi\}$
  - Draw an edge $e_i$ from the **<OperationName>** state to ***End*** State

- Set $\sigma_{in}$ / $\sigma_o$ = *Abort*
- Set guard condition $G(e_i) = \{True\}$
- Set reset condition $R(e_i) = \{\Phi\}$

- Create an edge $e_i$ from **<OperationName>** state to ***Check4Resume*** State
  - Set $\sigma_{in}$ = **<OperationName>Done** signal
  - Set guard condition $G(e_i) = \{True\}$
  - Set reset condition $R(e_i) = \{\Phi\}$

- Else if mission name is already there
  - Go to **Start** to look for the next order
- **End**

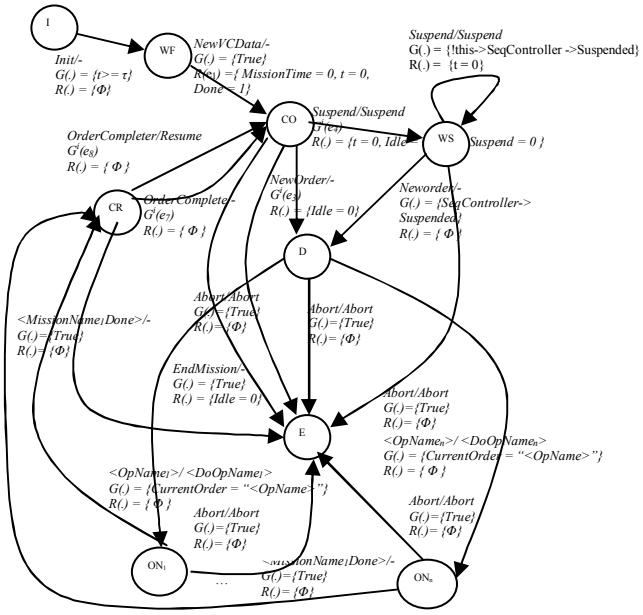


Figure 6: Timed Coordinator

### C. *Safety Coordinator synthesis*

Safety by definition is the freedom from danger, damage or risk. Thus the goal of a safety coordinator is to prevent the vehicle from taking actions, which might damage the vehicle. The safety coordinator monitors the different parameters involved in the operations ordered by the coordinators, the proper functioning of the components of the vehicle and the environment surrounding the vehicle. So a safety coordinator basically is an observer, which acts only when the operations lead to unsafe state. When the safety coordinator finds that a mission prompts execution of an unsafe action it tries to correct the action and make it safe. If the safety coordinator is not able to make the mission safe it aborts the mission. For example if a mission commands the vehicle to go to a depth of 500ft and the present safe depth is only 200ft the safety coordinator changes the depth to 200ft. If the safety coordinator is able to correct it the mission is carried out or else it aborts the mission. We here list a set of safety issues a safety coordinator should satisfy.

1. *Water depth safety* monitoring should check the altitude of the vehicle from the bottom of the sea and thus prevent the vehicle from hitting the bottom of the sea.

2. *Obstacle avoidance safety* should monitor the presence of obstacles, which might be other vehicles, or mountains under sea and prevent collision of the AUV with the obstacle. (This has not yet been implemented).

3. *Device functioning safety* should monitor the functioning of the different critical components, which constitute an AUV. Critical components are those components malfunctioning of which might lead to damage of vehicle or undesirable situation like AUV stuck at the bottom of the sea due to battery failure.

All these safety issues can be modeled as constraints within a hybrid system as has been done for the survey AUV built at ARL. The constraints are the guard conditions, which prompt the transition from one state to other depending upon the situation.

- Create locations $l \in L$ and name them as ***Start, Idle, CheckSafeties, LowAltitude*** and ***Abort***
- Draw an edge $e_0$ from ***Start*** state to ***Idle*** state (wait to get initialized)
  - Set event $\sigma_{in}$ = *Init/-*
  - Set guard condition $G(e_0) = \{t>= \tau\}$ where $\tau$ is a constant
  - Set reset condition $R(e_0) = \{\Phi\}$
- Draw an edge $e_1$ from ***Idle*** state to ***CheckSafeties*** state (wait for new vehicle command to arrive)
  - Set event $\sigma_{in}$ = *NewVCData/-*
  - Set guard condition $G(e_0) = \{t>= \tau\}$ where $\tau$ is a constant
  - Set reset condition $R(e_0) = \{\Phi\}$
  - *Implement the Voltage, Water depth and device safety algorithms.*
- Draw an edge $e_2$ from ***CheckSafeties*** state to ***LowAltitude*** state
  - Set event $\sigma_{in}$ = *AltitudeSafety/-*
  - Set guard condition $G(e_0) = \{t>= \tau\}$ where $\tau$ is a constant
  - Set reset condition $R(e_0) = \{\Phi\}$
  - *Implement the Water depth correction algorithm.*
- Draw an edge $e_3$ from ***CheckSafeties*** state to ***Abort*** state
  - Set event $\sigma$ = *Abort/Abort*
  - Set guard condition $G(e_0) = \{t>= \tau\}$ where $\tau$ is a constant
  - Set reset condition $R(e_0) = \{\Phi\}$
- Draw an edge $e_4$ from ***LowAltitude*** state to ***Abort*** state
  - Set event $\sigma$ = *Abort/Abort*
  - Set guard condition $G(e_0) = \{t>= \tau\}$ where $\tau$ is a constant
  - Set reset condition $R(e_0) = \{\Phi\}$
- Draw an edge $e_5$ from ***LowAltitude*** state to ***Checksafeties*** state
  - Set event $\sigma_{in}$ = *AltitudeOk/-*
  - Set guard condition $G(e_0) = \{t>= \tau\}$ where $\tau$ is a constant
  - Set reset condition $R(e_0) = \{\Phi\}$

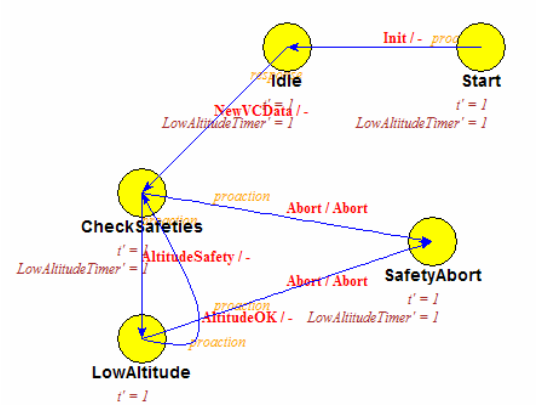


Figure 7: Safety Coordinator

The coordinators interact with each other to successfully execute a mission. The information flow of the interaction is as shown in Figure 8.

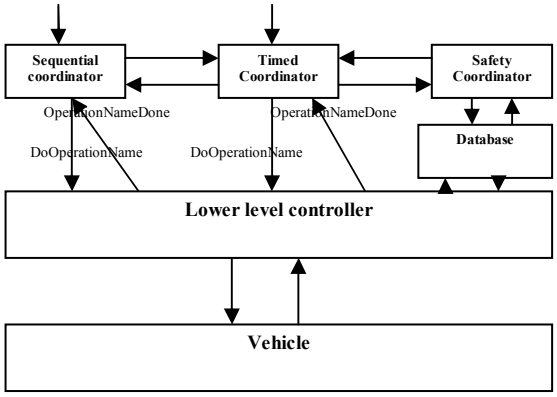


Figure 8: The Complete Structure

The present design is concerned with the successful execution of mission decomposed as sequence of operations ordered by the highest level controllers which we have automatically synthesized.

The analysis is provided based on the interactions between the modules shown in Figure 8 and the detailed modules of the sequential (Figure 5) and timed coordinator (Figure 6).

Both the coordinators are initialized first. During initialization the sequential coordinator establishes contact with the vehicle and the terminal from which mission orders are received.

When new order is received both the coordinators transition to the state at which they become ready to execute operations as a part of mission. If it's an untimed mission the sequential coordinator accepts the input and sends **<DoOperationName>** (Figure 8) to the lower level controllers. Once an operation is successfully executed the sequential coordinator receives **<OperationNameDone>** (Figure 8) from the lower level controllers**.** Then the SC considers the next operation in queue and passes control to the concerned lower level controller. If due to some malfunctioning the operation needs to be terminated an *abort* signal is received by the sequential coordinator from the lower level controllers involved in the mission. The sequential coordinator then broadcasts the *abort* signal (Figure 5) and terminates the execution of all other operations. If there are no more orders in the queue the SC checks for the status of the TC. If the TC is idle SC sends *EndMission* and transitions to the ***EndMission*** state (Figure 5).

If a timed mission is received then the timed coordinator checks whether the execution of the present mission needs the suspension of the sequential coordinator or not (these constraints are guard conditions on edges). If TC needs to suspend SC, TC sends the *suspend* signal to SC (Figure 6). If the mission which SC is executing is suspendable then SC synchronizes with the event *suspend* and transitions to the ***Suspend*** state (Figure 5). When SC is suspended TC sends the order as **<DoOperationName>** to the lower level controllers. The lower level controllers respond back with the **<OperationNameDone>** event to the TC when the operation is completed (Figure 8). TC then finds the next order in queue and either resumes the SC or keeps it suspended or keeps it unsuspended (Figure 6).

The safety coordinator keeps checking the parameters from the operations within a mission and sensor values of the AUV from the common database to safely execute a mission (Figure 8).

This way all the coordinators interact with each other and complete the execution of a mission order successfully. Next we discuss the propositions that should be satisfied by the coordinators for successful execution of a mission.

***Proposition 1:*** *(Given no Abort event) for all the orders there exist a response from the lower level controller which completes the mission successfully.*

Proof: *Proposition 1* can be reduced to the expression

$$\forall m \in M((\exists \sigma_i \mid H_i^j \xrightarrow{\sigma_i} H_k^p) \,\&\, (\exists \sigma_k \mid H_k^p \xrightarrow{\sigma_k} H_i^j)) \quad \text{--- 1.1}$$

Equation 1.1 states that for all missions there exist an event to pass control to the concerned commanded controllers as well as there exist an event to let the commanding controller know the completion of the mission.

From the coordinator synthesis algorithms we find that for the missions there exist a method to pass control from the higher level coordinator i.e. S.C or T.C. to the lower level coordinator which is to synchronize on common events <*DoOperationName*>. Thus we can express it as

$$\forall m \in M(\exists \sigma_i \mid H_i^j \xrightarrow{\sigma_i = <DoOperationName>} H_k^p)$$ where $i$ indicates the subsystem at level $j$, $k$ indicates the subsystem at level $p$, $j>p$ indicating that level $j$ is at a higher level than level $k$ --- 1.2

From the algorithms we find that each of the lower level coordinators respond back to a <*DoOperationName*> by a <*OperationNameDone*> event sent to the higher level controller.

$$\forall m \in M(\exists \sigma_k \mid H_k^p \xrightarrow{\sigma_k = <OperationNameDone>} H_i^j) \quad \text{--- 1.3}$$

Equation 1.2 and 1.3 together state that for each and every mission to be executed there exist an event to pass the control to the concerned controller as well as there exist a response which tells the higher level coordinator that the mission has

been successfully executed. Thus equation 1.1 holds so does *Proposition.1*.

***Proposition 2:*** *If the order is an Abort event it terminates the mission.*

Proof: The above *Proposition 2* can be reduced to the expression $\forall l \in L \forall \sigma = Abort \exists E \mid l \xrightarrow{\sigma = Abort} l_{final}$ --- 2.1

The expression states that for all locations belonging to a set of locations and for all events which are *Abort* events there exist an edge in which a transition occurs from the present location, the source to the target location, the final state. If the above expression holds for the coordinators synthesized by the algorithm then ***Proposition 2*** holds.

From the coordinator synthesis algorithm we find that there are statements which implement edges with *Abort* events from the ***<OperationName>*** locations to the ***Endmission*** location.

$$\textbf{<OperationName>} \xrightarrow{\sigma = Abort} \textbf{Endmission}$$

Thus equation 2.1 holds and so does *Proposition 2*.

***Proposition 3:*** *Timed as well as untimed missions can be successfully executed by the timely coordination between the Timed and Sequential coordinator.*

Proof: The above *Proposition 3* can be reduced to the expression $\forall m \in M \exists \sigma \mid (TC \xrightarrow{\sigma} SC)$ ---3.1.

The expression states that for all missions there exist a coordination event between the Timed Coordinator and the Sequential Coordinator for successful completion of both timed and untimed missions. If the above expression is satisfied by the coordinator synthesis algorithm then *Proposition 3* holds.

In the Sequential Coordinator synthesis algorithm we find statements dealing with creation of edges on value of Suspendable =1 and σ = *Suspend.* In the Timed Coordinator synthesis we find the implementation of edge $e_4$ which implements sending σ = *Suspend* to the Sequential Coordinator. The above statements reduce to

$\exists \sigma = Suspend \mid TC \xrightarrow{\sigma = Suspend} SC$ --- 3.2

The above expression states that there exist an event which allows TC to Suspend SC for execution of timed events.

From both the TC and SC synthesis algorithms we find statements implementing σ = *Resume* which helps in resuming the suspended SC. This statement reduces to the expression.

$\exists \sigma = \mathrm{Re}sume \mid TC \xrightarrow{\sigma = \mathrm{Re}sume} SC$ --- 3.3

From equations 3.2 and 3.3 we find that there exist methods of coordination between both the TC and the SC to execute timed as well as untimed missions. Thus equation 3.1 holds, so does the *Proposition 3*.

***Proposition 4:*** *Given a mission*

*(a) If no abort occurs during mission operation, then the mission will be successfully completed.*

*(b) If an Abort occurs during the mission, then the mission is terminated*

*(c) Timed and Sequential coordinator can coordinate among each other by suspending the other if required for execution of a mission.*

Proof: *Proposition 1 – 3* prove *Proposition 4*.

## V. RELATED WORK

Several approaches like game theory, supervisory control, and optimal control have been used to synthesize a controller. The supervisory control of discrete event system approach of Ramagde and Wonham [1] can also be said as the event feedback scheme. The plant generates events. The supervisor observes the events and then generates a control pattern based on a legal set of specifications. Other approaches have used state feedback control scheme [15]. The supervisor observes the plant states. At each step the supervisor generates a control pattern based on a given set of legal states to ensure no illegal state are reached.

The approach by Lygeros in [15], [12], [13], [14] is to design a hybrid controller by determining continuous control laws and conditions under which they satisfy the closed loop requirements. Then, a discrete design is constructed to ensure that these conditions are satisfied. Controller synthesis for a real time system is proposed by Asarin in [16]. The controller in [16] is synthesized based on a winning strategy for certain games defined by automata or timed automata. Another game theoretic approach proposed in [17] is used for constructing reliable controllers for arbitrarily large discrete systems. The controller is synthesized by finding a winning strategy for specific games defined by contracts. The discrete system model is an action system, and the requirement is a temporal property. The game reduces to a competition between, the controller, and the plant, which try to prevent each other from achieving their respective goals. If the synthesis is possible, that is, if the controller has a way to enforce the required property, the process ends with finding the winning strategy of the controller, by propagating backwards the computed precondition of the plant, with respect to that property. This technique guarantees the correctness of the derived program. Next we briefly discuss the interaction among some of the mission controllers to successfully execute a mission in our application to an AUV.

### *A. Modeling Mission Modules in TEJA*

TEJA allows the creation of a *system architecture* where all the modules required for a particular mission controller are instantiated and initialized, and their interactions are specified via an event dependency table which may be dynamically reset. Automatic code generation ensures that the real-time scheduling needs are met.

Figure 8-9 shows the hybrid automaton representation of the GPSFixer, Launch operation controller and steering behavior controller modeled using the Teja NP tool. On initialization the modules go to the ***Idle*** state from ***Start***. The GPSFixer (Figure 8) then goes to the surface, raises mast, updates the navigation system and then passes control to the Launch controller to lower the mast and then on the event *Launch* transitions to the ***ComeOffSurface*** state. The Launch controller then goes through its sequence of events shown in Figure 4 to lower the mast and lower the AUV below the surface of water. Then the Launch controller passes control to the GPSFixer controller on the event *LaunchDone*. The GPSFixer transitions to the ***Decide*** state where it decides whether to return back to the original

location before starting GPSFix mission or to just go to a particular depth. If the AUV needs to return to the original location the GPSFixer passes control to the Steer controller by outputting the event *Steer*. The Steer controller then executes its sequence of events (Figure 9). Once the AUV reaches the destined location the Steer controller passes control to the GPSFixer by outputting the event *SteeringDone*. The GPSFixer finally ends the mission by sending the event *GPSFixDone* to the TC or SC.

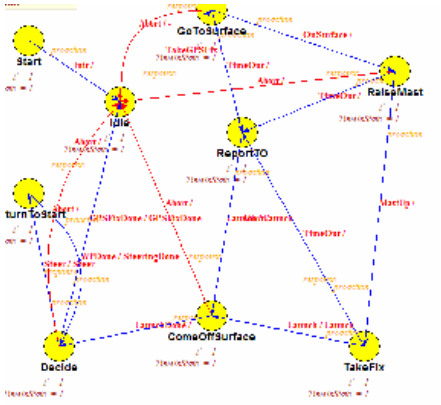


**Figure 9:** The GPSFixer Operation Controller

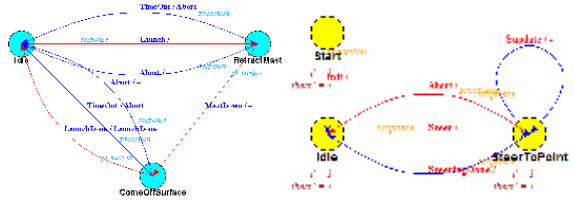


**Figure 10**: The Launch & Steering Controller modules

## VI. CONCLUSION

Synthesis of higher-level coordinators for AUVs whose mission controller is modeled in a hierarchical hybrid, model-based architecture is presented. This method of synthesis allows the generation of coordinators for any set of operation or behaviors and for any type of autonomous system (aerial, surface or underwater) modeled as shown in this work. This method of synthesis also guarantees the appropriate design of the coordinator modules to control the execution of sequence of timed or untimed operations/behaviors in a mission. The propositions prove that the algorithm satisfies the properties that should be satisfied by the coordinators. Future work involves implementing this proposed method using a software program for automated synthesis of coordinators.

## REFERENCES


[1] Ramagde, P.J., Wonham W.M.."The control of Discrete Event Systems", Proceedings IEEE, 1989, Vol. 77, No. 1, pp. 81-98.

[2] Holloway, L.E., Krogh, B.H., Giua, A.. "A survey of Petri Net Methods for Controlled Discrete Event Systems," Journal of Discrete Event Systems, 1997, Vol. 7, No. 2, pp. 151-190.

[3] Datta N. Godbole, John Lygeros, and Shankar Sastry, "Hierarchical Hybrid Control: a Case Study", LNCS 999, June 1995.

[4] R. Alur, T. A. Henzinger, and E. D. Sontag, Eds., Hybrid Systems III. New York: Springer-Verlag, 1996, vol. 1066, Lecture LNCS.

[5] www.teja.com

[6] www.uppaal.com

[7] R. Alur, T. A. Henzinger, G. Lafferriere, And George J. Pappas *Discrete Abstractions of Hybrid Systems*. Proc. of the IEEE, July 2000.

[8] T. A. Henzinger, P-H Ho, and H W-Toi, 1995, "A user guide to HyTech", *Proc. of the First International Workshop on* TACAS '95, LNCS 1019, Springer-Verlag, 41-71.

[9] OpenGL programming guide – Redbook version 1.4

[10] S. Tangirala, R. Kumar, S. Bhattacharyya, M. O'Connor, and L. E. Holloway,"Hybrid-Model based Hierarchical Mission Control Architecture for Autonomous Undersea Vehicles", American Control Conference (ACC), June 2005.

[11] M.O'Connor, S.Tangirala, R.Kumar, S. Bhattacharyya and L.E. Holloway, "Hybrid-Model based Hierarchical Mission Control Architecture for Autonomous Undersea Vehicles: Verification", ACC 2006, Minneapolis, MN

[12] Lygeros, J.; Godbole, D.N.; Sastry, S. "A game-theoretic approach to hybrid system design", IN: *Hybrid Systems III. Verification and Control*, New Brunswick, NJ, USA, 22-25 Oct. 1995). Edited by: Alur, R.; Henzinger, T.A.; Sontag, E.D. Berlin, Germany: Springer-Verlag, 1996. p. 1-12.

[13] C. Tomlin, G. Pappas, and S. Sastry, ``Conflict resolution for air traffic management: A case study in multi-agent hybrid systems," tech. rep., UCB/ERL M97/33, Electronics Research Laboratory, University of California, Berkeley, 1997. To appear in the IEEE Transactions on Automatic Control, Special Issue on Hybrid Systems, April 1998.

[14] Lygeros, J.; Tomlin, C.; Sastry, S. "Multiobjective hybrid controller synthesis", IN: Hybrid and Real- Time Systems. International Workshop, *HART'97. Proceedings*, Grenoble, France, 26-28 March 1997). Edited by: Maler, O. Berlin, Germany: Springer-Verlag, 1997. p. 109-23.

[15] J. Lygeros, D. N. Godbole, and S. Sastry, "Verified hybrid controllers for automated vehicles," IEEE Trans. Automat. Contr., vol. 43, pp. 522–539, Apr. 1998.

[16] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid System II*, volume 999 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995

[17] Ralph-Johan Back, Cristina Cerschi Seceleanu, Contracts and Games in Controller Synthesis for Discrete Systems, 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04) 05 24 - 05, 2004 Brno, Czech Republic

[18] S. Bhattacharyya, R.Kumar, S.Tangirala, M.O'Connor, and L.E. Holloway, "Animation/ Simulation of Hybrid-Model based Hierarchical Mission Control Architecture for Autonomous Undersea Vehicles", ACC 2006,Minneapolis, MN