View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by VBN

**Aalborg Universitet**

AALBORG UNIVERSITY
DENMARK

# The SOPHY Framework

Laursen, Karl Kaas; Pedersen, Martin Fejrskov; Bendtsen, Jan Dimon; Alminde, Lars

*Publication date:*
2005

*Document Version*
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

*Citation for published version (APA):*
Laursen, K. K., Pedersen, M. F., Bendtsen, J. D., & Alminde, L. (2005). The SOPHY Framework: Simulation, Observation and Planning in Hybrid Systems. Aalborg: Department of Control Engineering, Aalborg University.

**General rights**
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

? Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
? You may not further distribute the material or use it for any profit-making activity or commercial gain
? You may freely distribute the URL identifying the publication in the public portal ?

**Take down policy**
If you believe that this document breaches copyright please contact us at vbn@aub.aau.dk providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from vbn.aau.dk on: April 25, 2017

# The SOPHY framework: Simulation, Observation and Planning in Hybrid Systems

Karl Kaas Laursen, Martin Fejrskov Pedersen, Jan Dimon Bendtsen, Lars Alminde
*Department of Control Engineering, Institute of Electronic Systems*
*Aalborg University, Denmark*
*Email: {karl, mfpe01, dimon, alminde}@space.aau.dk*

## Abstract

*The goal of the Sophy framework (Simulation, Observation and Planning in Hybrid Systems) is to implement a multi-level framework for description, simulation, observation, fault detection and recovery, diagnosis and autonomous planning in distributed embedded hybrid systems. A Java-based distributed, hybrid simulator is implemented to demonstrate the virtues of Sophy. The simulator is set up using subsystem models described in human readable XML combined with a composition structure allowing virtual interconnection of subsystems in a simulation scenario. The performance of the simulator has shown to be very dependent on the way its distributability is utilised revealing both the limitations and strengths introduced by delegating computation tasks in a distributed architecture.*

## 1. Introduction

Recently, a revived interest in the field of artificial intelligence (AI) has surfaced in research areas without any prior history in AI. One such scientific territory is the area of space exploration, which is starting to see the possible advantages of a more autonomous way of system operation. As of today, unmanned spacecraft are completely isolated and self-contained systems without physical human contact, remotely controlled via round-the-clock teleoperation by specialists. Such personnel may have to spend six to seven hours a day devising operational procedures and flight plans and the rest of the time on actual operation, data collection, maintenance, fault analysis and error handling, making it a full time job to operate the remote system [7].

Controlling a highly complex system like a spacecraft from a distant location brings about the inevitable risk of unrecoverable human errors resulting in mission goal reduction or even mission failure due to the extremely restricted access to the system itself. If the spacecraft were able to handle some, if not all, of the tasks required to fulfill its mission goals, the required human intervention would be lessened, leading to less stress on the mission personnel and less likelihood of failure. To achieve this, a certain level of on-board intelligence on the remote platform is required. The system must be able to sense its surroundings, simulate likely behavior in response to various control actions, and choose the optimal actions according to relevant criteria, such as achievement of mission goal, energy consumption, collision danger etc.

These considerations are obviously not only valid for spacecrafts, but indeed for any autonomous system [2]. This motivates the development of a structured framework for constructing systems that are able to operate without human intervention, both for the purposes of control design, but also to facilitate integrated design and testing of autonomous systems. Such a framework must necessarily involve methods for simulating and observing the system, as well as easy implementation of control and planning algorithms.

Since autonomous systems very often operate in different modes in order to carry out their tasks, the framework must be able to support formulations of systems involving both continuous and discrete dynamics, i.e. *hybrid systems*. For example, a mobile exploration robot on a mission to gather rock samples might drive to a specified waypoint using a tracking controller (see e.g., [3]). Then, at the waypoint, it might activate a robot arm for gathering up the rock sample, requiring an entirely different controller to be active (see e.g., [4]). Instead of attempting to describe every part of this complex system as one large state space model, it is more logical to model and control each subsystem separately and let the framework handle the composition and simulation. Furthermore, it must be possible to describe the subsystems using a simple formalism that lends itself to easy interpretation on and exchange of data between computers for the sake of distribution of tasks.

The SOPHY (Simulation, Observation and Planning in HYbrid systems) framework is a common framework for hybrid model description and composition that intends to address these issues. It utilizes object-oriented design and implementation methodologies and is intended to be fully

distributed. In this paper, based on the master's thesis [5] founding SOPHY, we present the initial design and implementation of a fully distributed simulation tool within this framework, which is able to simulate distributed hybrid systems, including interactions between them. In the rest of the paper, we first present an overview of the hybrid systems considered in the simulator as such, followed by a brief description of the framework and software structure. Then we present a distributed simulation example and end with a few concluding remarks.

## 2. Hybrid Systems

Hybrid systems are systems involving both continuous and discrete dynamics. An example of a hybrid system comprised of two hybrid subsystems is shown in Figure 1. Each subsystem consists of two discrete *locations* each with its own continuous dynamic equations and two *transistions* to allow location changes. Furthermore, the locations in subsystem 1 has *invariant sets* described by state-dependent inequalities $i_1$ and $i_2$.

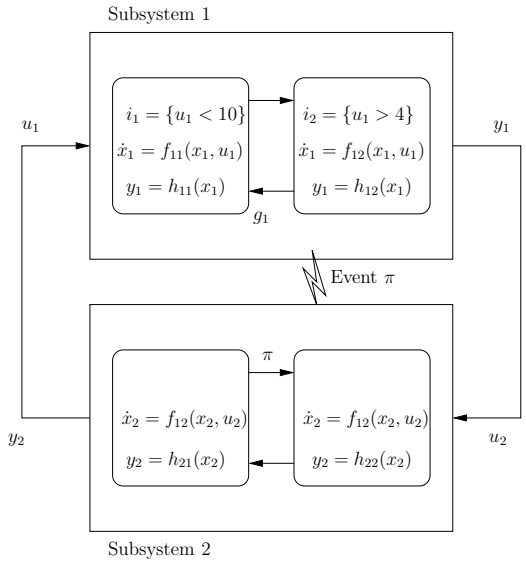


**Figure 1. Two interconnected hybrid subsystems each with two locations, two sets of invariants and no exogenous inputs.**

The behavior in each discrete location can be described as an ODE governing the dynamics at the location, $f_{v_s}$, an output equation $h_{v_s}$ and an invariant map, $i_{v_s}$, describing a powerset of states and inputs which gives the domain of the location:

$$\dot{x}(t) = f_{v_s}(x(t), u(t)) \quad y = h_{v_s}(x(t), u(t)) \quad (1)$$
$$i_{v_s} : v_s \rightarrow 2^{\mathcal{X}^n \times \mathcal{U}^m} \quad (2)$$

where $x \in \mathcal{X}^n \subseteq \mathbb{R}^n$ is a vector of continuous states, $u \in \mathcal{U}^m \subseteq \mathbb{R}^m$ are exogenous inputs, $y \in \mathcal{Y}^p \subseteq \mathbb{R}^p$ are outputs and $t$ is the time. $\mathcal{X}^n, \mathcal{U}^m$ and $\mathcal{Y}^p$ are state, input and output spaces. $v_s$ is discrete location $s$.

As time progresses the system state evolves within a single location, until an external event occurs or the invariant inequality is unsatisfied, i.e. $(x, u) \notin i_{v_s}$. At this point, a *transition* can be triggered, causing the system to shift to the other set of continuous dynamics, possibly after a reset of the continuous states. If a location has more than one possible transition to other locations the transitions can be *guarded* by a *guard set* which, like the invariant set, is a powerset of states and inputs. In order for a transition to be enabled, the guard set must be satisfied.

### 2.1. Events

In discrete event systems events and transitions are closely coupled: a transition can only occur when triggered by the occurrence of an event. This is due to the fact that all dynamics in a discrete event system is caused by the stochastic or deterministic occurrence of events. This is, however, not the case with hybrid systems and for this reason events are explicitly defined separately from transitions: events are emitted from a subsystem if and only if a certain transition has occurred. In an Event-Open Hybrid Automaton an event originates deterministically internally to a system as a *result* of a transition. The intended use of events according to the above description is thus as an inter-(sub)system communication infrastructure.

### 2.2. Transition types

We consider three types of transitions:
**Event transitions;** Event transitions are those transitions associated to events (if any), which are triggered instantaneously when a subsystem receives an event, if the transition is enabled. If the associated transition is not enabled, the event is simply discarded.
**Rate transitions;** Rate transitions are the equivalent of events as defined in stochastic timed discrete event systems: Rate transitions occur stochastically according to a Poisson process with the Poisson rates $Q$. The main difference between the rate transitions defined here and the events in a normal DES or a PDMP [1] (called "switching transitions") is that the rates may be dependent on the states and inputs.
**Invariant transitions;** Invariant transitions occur when the state crosses the invariant boundary. If one or more transitions are enabled or become enabled when the invariant boundary is crossed, one of these transitions occurs instantaneously. If no transition is enabled, no transition occurs. Invariant transitions are common in hybrid systems defini-

tions but in many cases these are defined to be triggerable even if the guard and/or the invariant set in the destination location are not satisfied.

### 2.3. Synchronized Guards

The parallel compositional equivalent of the aforementioned events is synchronized guards. When simulating hybrid systems composed of several subsystems, it is necessary to employ a signaling mechanism where the systems, containing a guard that must be synchronized, set a "flag". Upon the registration of the flag from all synchronized guards, the transition is enabled in all subsystems and the transition can be triggered by one of the three above mentioned transition methods.

These concepts were formulated in an object-oriented framework by means of Unified Modelling Language (UML). This formulation was then used in the design and implementation of the SOPHY simulation tool.

### 2.4. UML Description

UML is a powerful tool to describe connections between entities of different types. This also applies to the elements of the hybrid systems definition which can be seen as a collection of classes with certain associations each with a specified multiplicity. Using a class association diagram from the UML family it is possible to graphically illustrate how locations, transitions, states, guards, etc, are associated to each other, including the multiplicity of the output of mappings. The hybrid system model structure proposed in this paper can be illustrated as in Figure 2 and it is described in detail in [5].

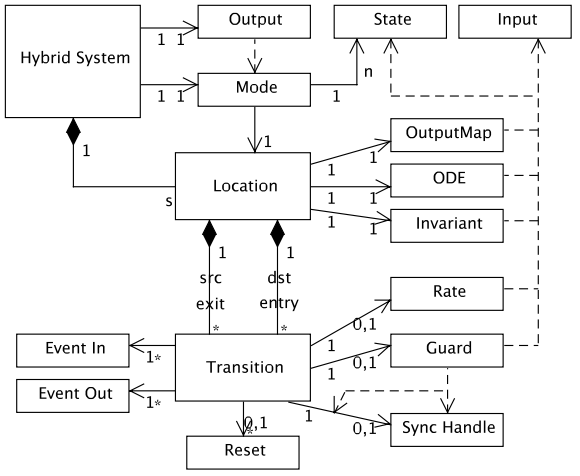


**Figure 2. UML description of hybrid systems.**

## 3. Framework

As outlined in the introduction, SOPHY is a framework architecture aimed at implementing advanced autonomy in systems that can be described as hybrid systems; in general, this concerns complex systems often composed by multiple subsystems. The full architecture of SOPHY is defined and a hybrid simulation component has been implemented and tested. The framework architecture is outlined in Figure 3 and the following describes the components in greater detail.

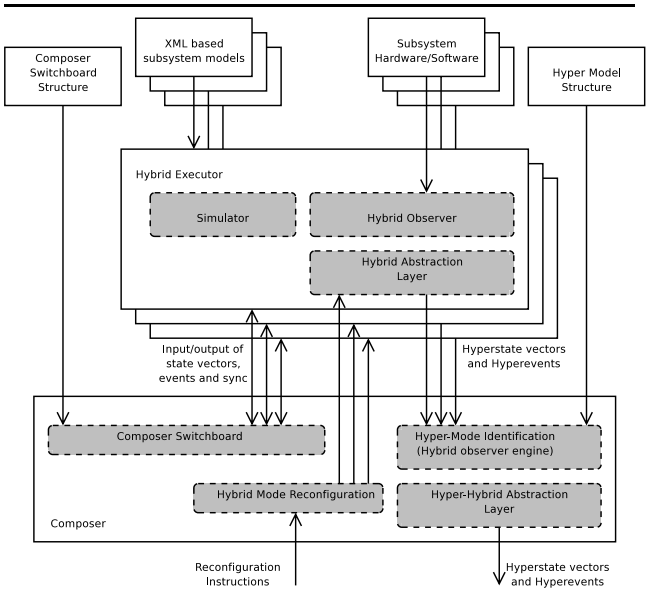


**Figure 3. The SOPHY framework.**

### 3.1. Input Models

A key architectural point in the framework is that it is *declarative* meaning that the user should only be concerned about describing a system and not about implementing specific controllers and observers for the system. This is a major break from traditional thinking in automatic control.

To facilitate this the only human inputs to the framework are hybrid models described in human readable XML files. On Figure 3 these are indicated as rectangles at the top constituting hybrid models of the different subsystems in the system and a file describing the interconnections between the different subsystems. An optional file describes the system in the "hyper-domain" (to be explained in the following sections).

### 3.2. Hybrid Executor

The Hybrid Executor (HE) is the architectural element in the framework that is responsible for simulation, obser-

vation and control of a single subsystem. Presently, a general purpose simulator that can simulate hybrid systems (as described in section 2) is implemented. The framework anticipates that also a general hybrid observer is implemented together with a general purpose hybrid controller. General purpose control and observation is envisioned to be implemented using, for example, Unscented Kalman Filtering (UKF) for observation and Model Predictive Control (MPC) for control. Both of these techniques are suited to operate in a declarative environment with a system model as their only input and without the need for comprehensive manual tuning.

Finally, each HE contains an element dubbed ”Hybrid Abstraction Layer (HAL)”. For some applications there may be a high degree of high-level coordination requiring the knowledge of most internal states of the subsystems and in other applications each subsystem may be almost detached from the rest of the system. The HAL is responsible for implementing the degree of transparency that is required for each subsystem as dictated by the model.

### 3.3. Composer

The Composer coordinates the information flow between the attached HEs and contains elements for high-level control and supervision. The switchboard connects data-channels on a subscription basis, meaning that any HE can subscribe to an output of another HE as described in the ”Composer Switchboard Structure” XML file. The ”Hybrid Mode Reconfiguration” engine is responsible for goal oriented coordination and control of the HEs. A system similar to the Planner/Scheduler (PS) of the Deep Space 1 project (DS1) [6] is envisioned to fulfill this role. Finally the ”Hyper-Mode Identification” and ”Hyper-Hybrid Abstraction Layer” are included in the framework as a mean to facilitate yet another level of Composers if the grouping of Composers is desirable in very complex systems. As an example, a Composer might compose the HEs describing the engine of a car, and another Composer might compose the HEs describing the driver. Then another, higher-level, Composer might compose the engine Composer with the driver Composer to create a “car with driver” object.

### 3.4. Comparison to Deep Space 1

The design of the SOPHY framework was motivated by the design of the autonomy system for the NASA DS1 space craft. However, there are many architectural differences. Firstly, SOPHY, like DS1, is based on declarative models, but in SOPHY these models only enter at one level in contrast with DS1 where each engine in the architecture has its own model of each subsystem. This means that it is more time efficient to write models for SOPHY and the risk of ambiguity is eliminated.

Secondly, SOPHY is built to support models of hybrid systems from the bottom up, whereas in DS1 low-level models were Discrete Event Systems (DES) and high-level models were DES with augmented simple algebraic models for resource consumption. SOPHY gathers the models in hybrid systems for more realistic model implementation for the sake of versatility, reliability and framework operation speed.

## 4. Software

A key issue of the SOPHY framework is its distributed, object oriented software implementation. It is implemented in the Java 1.5 language and all data traffic in the distributed framework uses the TCP/IP protocol. This allows the components to be distributed as depicted in Figure 4 where a computer is appointed the role of composing the activities carried out on a number of servers each hosting a Hybrid Executor. The core and infrastructure of the SOPHY software package, together with a hybrid simulation component, is implemented and tested.

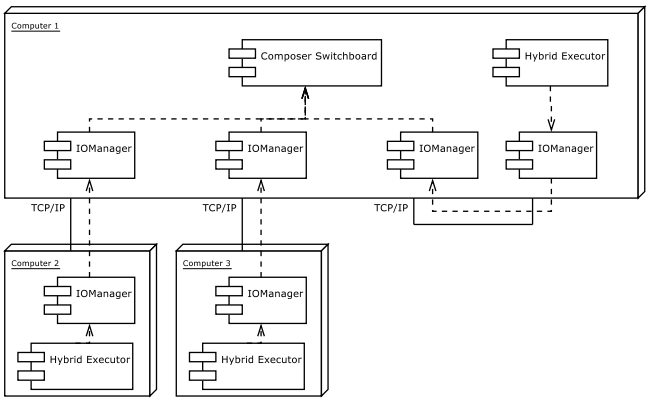


**Figure 4. The deployment of the software components.**

### 4.1. Simulator

The main architecture of the simulator is designed with the designated goal of making it modular and versatile such that different simulation algorithms can be implemented and chosen to fit any given simulation job at hand. This is achieved mainly through the delegation of the simulation task onto three sub-components: Simulator, TransitionGenerator and Solver. The relationships between the main

classes involved in a simulation scenario are depicted in the class diagram in Figure 5.

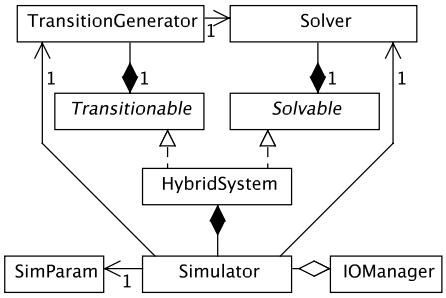


**Figure 5. The classes essential to the simulator core.**

The responsibilities of the Simulator are to communicate with the composer switchboard via the IOManager, decide when to investigate the hybrid system for possible transitions and to get output from HybridSystem after each simulation step and ship it out to the switchboard and possibly to a PlotViewer component. The TransitionGenerator makes transitions in the HybridSystem via its *Transitionable* interface. The Solver propagates the continuous states of the hybrid system model via the *Solvable* interface to HybridSystem. The Solver and TransitionGenerator sub-components are the most essential parts in that they compose the algorithmic behaviour of the simulator. Therefore, these are made interchangeable by utilizing a variant of the object oriented design pattern *bridge design* which allows for runtime change of simulation algorithms without program recompilation. Currently implemented solvers include Euler, Heun, RK, RKF and ABM.

## 5. Simulation Example

In order to demonstrate the feasibility of the distributed, hybrid simulator, an example of a multi-body hybrid dynamic system is simulated. The example system consists of five hovering vehicles (hovercraft) which are actuated in two dimensions by force-linear thrusters. Each of the vehicles has an associated hybrid controller to control the force generated by the thrusters. The inputs to the controller are the velocity of its hovercraft and the position of all other vehicles and obstacles such as the walls of the confinement where the craft are operating. The output is a force reference to the thrusters of the hovercraft.

### 5.1. Hovercraft Dynamics

The motion of the hovercraft is governed by the simple linear rigid body dynamics in the described by equation 4.

$$\begin{aligned} \begin{bmatrix} \dot{v} \\ \dot{p} \end{bmatrix} &= \begin{bmatrix} -\mu/m & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ p \end{bmatrix} + \begin{bmatrix} 1/m \\ 0 \end{bmatrix} \cdot u \\ y &= \begin{bmatrix} v \\ p \end{bmatrix} \end{aligned} \tag{3}$$

where $v$ is velocity, $p$ is position, $m$ is the mass of the craft, $\mu$ is viscous friction, $u$ is the thruster force equal to the controller input and $y$ is the output.

### 5.2. Hybrid Controller Automaton

The hybrid controller operates in two discrete locations: *nominal* and *avoid*. In *nominal* the controller is a speed controller that simply tries to keep the vehicle at a constant speed reference in the direction it is moving. The domain of *nominal* is the set of vehicle positions where the minimum distance to every other vehicle and obstacle is greater than 2 m. If the distance is less than this, the controller makes a transition to the *avoid* location where a proportional controller takes over. This uses the inverse of the distance to nearby objects as an error signal to generate a control output resulting in a force directed away from these objects. The domain of *avoid* is the set where the minimum distance to all objects is less than 4 m. The controller is described as a hybrid automaton in the simplified equation 4.

$$\begin{aligned} V &= (v_1, v_2) = (nominal, avoid) \\ E &= (e_1, e_2) = ((v_1, v_2), (v_2, v_1)) \\ x &= (x_1) \\ u &= (v, p_1, \ldots, p_5) \\ y &= (y_1) \\ f(v,x,u) &= \begin{cases} (3 - \|v\|)/33, & \text{if } v = nominal \\ 0, & \text{if } v = avoid \end{cases} \\ h(v,x,u) &= \begin{cases} 333 \cdot (3 - \|v\| + x_1) \cdot \frac{v}{\|v\|} \\ 1000 \cdot \sum_n \frac{p_1 - p_n}{\|p_1 - p_n\|} \end{cases} \\ i(v,x,u) &= \begin{cases} \{min(\|p_1 - p_2\|, \ldots, \|p_1 - p_5\|) > 2\} \\ \{min(\|p_1 - p_2\|, \ldots, \|p_1 - p_5\|) < 4\} \end{cases} \end{aligned} \tag{4}$$

### 5.3. Simulation Results

A simulation of five hovercraft and five hybrid controllers has been carried out using the distributed simulator. In this case, there is a total of ten subsystems (five craft plus five controllers), and the simulation is distributed to five computers. Figure 6 depicts the trajectories of the vehicles showing how the hybrid controller makes them stay within the confinement while avoiding collisions.

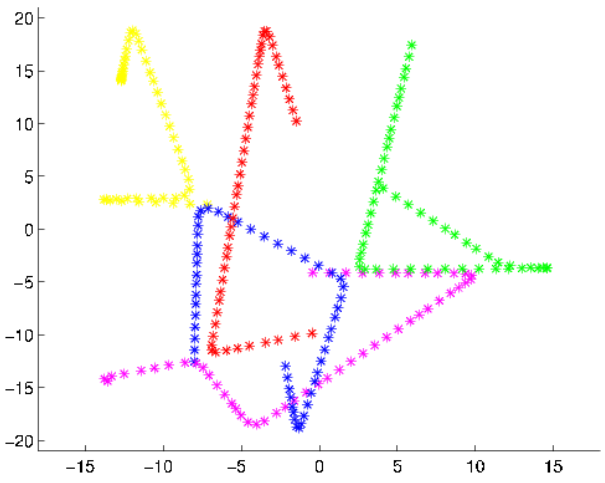


**Figure 6. Simulation of five hovercraft with hybrid speed and avoidance controller.**

### 5.4. Benchmark

A number of tests have been carried out in order to establish an empirical measure of the performance of the simulation framework against an established alternative in classical simulation, for example using Matlab. The benchmarks were done using ten computationally heavy subsystems containing a model of spacecraft rigid body dynamics and kinematics. The following observations were made:

1. When running all simulators on a single host, the time used to set up the simulation (start the Java Virtual Machines, build and distribute the ten models, send simulation parameters etc.) was approximately 15 s. The time for the actual simulation was 20 s. The computer load average was 1.1, approximately equivalent to one fully utilized CPU.
2. When running each of the ten simulators and the composer switchboard on a total of elleven hosts, the setup time was approximately the same as above (15 s). The total simulation time was approximately 9 s, and the load average on the HE hosts was app. 0.1, equivalent of a 10% utilization of a single CPU.
3. When doing a single-host simluation of a single subsystem without using network, the simulation time boils down to 1.4 s using an Adams-Bashforth-Moulton solver algorithm.
4. Implementing and simulating the single rigid body model in Matlab using a self-implemented Runge-Kutta algorithm in an .m-file yields a total simulation time of approximately 90 s. Using the fixed-step version of the ODE45 function, however, yields a total simulation time of approximately 0.5 s.

## 6. Conclusion

This paper presents a framework for simulation, planning and control of autonomous systems, along with an initial implementation of an automated, distributed simulation tool for hybrid systems. The long-term aim of the work presented here is to implement a multi-tier framework for description, simulation, observation, fault detection and recovery, diagnosis and autonomous planning in distributed embedded hybrid systems. The idea was to facilitate capture of the behavior of real-world systems by allowing not only differential equations, locations, transitions, invariants and guards, but also stochastic elements to be modelled into a system description as well as the exchange of discretely emitted events between subsystems. In addition, a new definition of synchronous guards across subsystem boundaries has increased the versatility of the hybrid systems definition.

A hybrid simulation tool has been developed and implemented in the Java programming language. The simulator is designed to work in multiple embedded systems with no guaranteed human access to the actual computing devices which is achieved through the use of a multi-purpose server component called the SophyServer. The simulator is set up using subsystem models described in human readable XML combined with a composition structure, allowing virtual interconnection of subsystems in a simulation scenario, which is also described in XML. Tests showed that it is comparable in terms of performance (speed and accuracy) with Matlab's ODE45 differential equation solver.

## References

[1] Elena De Santis and Maria D. Di Benedetto and Stefano Di Gennaro and Giordano Pola. *Hybridge Work Package 7, Hybrid Observer Design Methodology*, August 2003.

[2] M. L. Bujorianu and J. Lygeros and W. Glover and G. Pola . *Hybridge Work Package 1, A Stochastic Hybrid System Modelling Framework*, May 2003.

[3] B. d.-N. Benoit Thuilot and A. Micaelli. Modeling and feedback control of mobile robots equipped with several steering wheels. *IEEE Transaction on Robotics and Automation*, 12(3):375–390, June 1996.

[4] S. H. et al. *The Rocky 7 Rover: A Mars Sciencecraft Prototype*.

[5] K. K. Laursen and M. F. Pedersen. Online composition and distributed simulation of hybrid systems. Master's thesis, Aalborg University, Denmark, 2005.

[6] N. Muscettola, B. Smith, et al. On-board planning for new millennium: Deep space one autonomy. *IEEE Aerospace Conference, Aspen, CO*, 1997.

[7] Thomas Oberst. *Restless rovers demand long hours from CU's 'Martians'*, 2004. www.news.cornell.edu/Chronicle/04/11.11.04/MarsLab.html.