15th Triennial World Congress, Barcelona, Spain

TOWARDS AN INTEGRATED CONCEPTION OF HYBRID DYNAMICAL SYSTEMS

Valérie Roy *,[1] **Nadia Maïzi** *

* *Ecole des Mines de Paris, Centre de Mathématiques Appliquées*
2004 rte des lucioles BP 93, 06902 Sophia-Antipolis cedex France
{*valerie.roy, nadia.maizi*}*@sophia.inria.fr*

Abstract: Hybrid dynamical systems are composed of continuous-time dynamical parts, mixed with event-driven parts. Most of the time, both parts are designed separately using specific techniques of each domain, and integrated *a posteriori* in an application-specific manner. This approach is restrictive in that it does not exhibit a hybrid global model of the designed system, that would though be required for analysis and behavior-checking to take place. In this paper, we discuss and illustrate our approach of hybrid systems modeling, that is based on the obvious statement that both domains (dynamical and event-driven) must be clearly considered in an integrated manner from the very beginning of the design.
In our example, we exhibit a draft formal framework for hybrid system modeling, that would allow for verification techniques. For that purpose, we take advantage of the recently developed techniques and tools, in both areas. The numerical computation laboratory - matlab - that we chose for the dynamical system part design, fits perfectly with our goals. But the reactive synchronous language chosen - Esterel - , if it actually fits with the event-driven part specification, exhibits some weaknesses when dealing with data and values, that are needed when interfacing both parts together.

Keywords: dynamical systems, control-dominated reactive synchronous languages, hybrid systems

1. INTRODUCTION

Hybrid dynamical systems involve dynamics with continuous or discrete time (physical phenomena) and events. Both control theory and computer science communities show a great interest for these systems. On the one hand, industrial applications deal, more and more often, with problems of increasing complexity that should be solved by a global, thus hybrid, approach. On the other hand the drastic improvement of computer power opens new perspectives for the implementation of specialized algorithms well adapted to fit with real-sized problems. Improvements in hardware and software technologies are at the base of the extension, evolution, democratization and increasing functionalities of computer systems. Applications in the hybrid dynamical systems field - most of the time embedded - are critical and need bug-free implementations. It becomes necessary to provide with an environment, appropriate for formal modeling of such systems. By involving verification techniques, this environment will then allow for safer design.

In this paper, we first describe dynamical systems from a hybrid standpoint. We discuss the fact that some classical families of dynamical systems clearly exhibit a hybrid behavior, and that being considered as natively hybrid, they could thus

[1] Partially supported by ANVAR

benefit from any result obtained within the hybrid system research field.

Then we discuss of event-driven systems from the reactive concept standpoint. We briefly present the language, the hypothesis done to obtain a valid model - like atomicity - and the semantics needed to compile a program into the formal model of finite automaton or net-list.

After this overview of the two domains, we present an experiment of hybrid system design. As a trade-off, it takes advantage of a pure event-driven formalism in order to take into account behavior and interface of dynamical sub-systems, while preserving the semantics of the reactive language. Interactions between both phenomena are thus considered at the very beginning of the implementation in a closely integrated manner.

The last section describes an example which illustrates the interest of our approach. In this example, hybrid system modeling is done using specific tools and techniques coming from each domain but integrated in a formal way that provides us with a global hybrid model.

The conclusion outlines some restrictions coming from the selected reactive language, and suggests some improvements that can be envisioned.

2. TOWARDS A HYBRID UNDERSTANDING OF DYNAMICAL SYSTEMS

Within classical control theory, during a system design phase, the dynamical modeling step aims at describing the physical phenomena coming from mechanics, biology, electricity, ... From this model, the control theory can be applied; the analysis phase then uses a standard formalism, among : finite-dimensional linear dynamical systems, infinite-dimensional, non-linear, ... The analysis and study phases of these systems (dimensioning, stability study of the system, control, optimization ...) are the different issues that control theory is about.

Among the specialized arrays of control theory, some of them - such as e.g. variable-structure systems and sliding modes(Utkin, 1977), theory of singular perturbation, discrete events system(F. Baccelli and Quadrat, 1992), - naturally exhibit a hybrid behavior. Thus any of these research fields could immediately benefit from any wide-purpose result that could be established in the hybrid systems arena.

In the current hybrid context, both aspects (continuous and event-driven) are considered separately and brought together in an *ad hoc* manner. For a genuine study of these systems from a hybrid standpoint, there is a need for an earlier and more thorough consideration of the interactions between both sides.

On the other hand, the research fields of control theory (quadratic linear control, H^∞ control, game theory, ...) are likely to be subject of further improvements in the hybrid area.

For numerous years, the design of continuous systems has been able to take advantage of the methodological and modeling framework that came out with the numerical computing laboratories, from both industry (matlab) and research (scilab). These laboratories, widely used among the industry, are becoming references of the domain, or even maybe standards. But if they happen to include tools for addressing the event-driven parts, - e.g. the stateflow toolbox in matlab - this is never done in an integrated fashion, nor does it allow for a formal modeling of the hybrid aspects. Anyway, the related implementations are most of the time of a far lower level of quality, as compared with their numerical counterparts.

2.1 *How does a dynamical system get hybrid ?*

Let us consider a continuous system described by its evolution law

$$\dot{x}(t) = f(x(t), u(t), t)$$

Let us also consider a set of events e_i, generated outside the system. The event-driven context may interfere with the continuous system in one of the ways formalized below :

(1) notify a change in trajectory, among a predefined set :
$\{e_l, t_l\}_l$ change in trajectory at time t_l at occurrence of event $e_l \Rightarrow \dot{x}(t) = f_l(x(t), t)$

(2) notify a change in initial conditions, resulting in re-computing the trajectory :
$\{e_j, t_j, x_j\}_j$, x_j change in initial conditions at time t_j at occurrence of event $e_j \Rightarrow \dot{x}(t) = f(x(t), u(t), t)$

(3) notify a change in the control law, resulting in re-computing the trajectory :
$\{e_k, t_k, u_k\}_k$, u_k change in the control law, at time t_k at occurrence of event $e_k \Rightarrow \dot{x}(t) = f(x(t), u_k(t), t)$

This list might be non exhaustive because we did not consider all the possible perturbations affecting a dynamical system.

Our purpose is to formalize a hybrid dynamical system as a global loop. We imagined a simple implementation of these concepts, so as to obtain a tool allowing for an analysis of these systems from the control theory standpoint (analysis, dimensioning) and from the software engineer standpoint.

3. OUR UNDERSTANDING OF EVENT-DRIVEN SYSTEMS

An event-driven system keeps on :

- waiting for events originating from its environment,
- then in turn producing events to the outside world.

They are also known as reactive systems. The design of such systems was the subject of numerous researches, done for years, and that came out with various models, methods and tools such as grafcet, Petri net, statemate, reactive language. We will focus on the latter approach.

3.1 *The reactive approach of event-driven systems*

The following section accurately describes what we mean by event-driven systems taken as control-dominated synchronous reactive systems.

The outside environment provides the event-driven system with an input event (a set of signals), then activates the reactive system, that in turn reacts by emitting an output event. This sequence - input, activation, output - is called a reaction, and matches in this model an **instant**. The reactions rhythm is driven by the environment and the sequence of these instants describes the basic abstract clock of the system. Within this model, time is polymorphic, i.e. any signal is considered the same way, physical time has no special privilege : signals can be METER, a SECOND or a BUTTON_ON.

Event-driven systems are naturally composed of concurrent sub-systems, communicating together and with the outside, each of them implementing a part of the specifications. Their design methods must thus allow concurrency and modularity. Signals are the abstract communication media between environment and system, system and environment and between sub-systems. Communication is non-blocking and is done by synchronous broadcast (like radio). When a signal is emitted, it is seen everywhere inside the program : any concurrent sub-system has a consistent view of signal status (present/absent).

From these requirements, some languages have been designed. They contain various operators allowing us to wait, emit and test signals, to preempt sub-program, to put sub-program in sequence or in parallel, to loop, to help organize a program as a hierarcical and/or modular entity.

Many hypothesis are done for these systems to present a consistent behavior. A reactive system must have enough time to end its current reaction before the input of the following reaction happens. This hypothesis guarantees that a reactive system never loses an input. So these systems are by definition real-time systems. Another hypothesis, named atomicity, prevents an input signal to happen in the middle of a reaction. The last requirement for our reactive system is to present a deterministic behavior.

An operational semantics has been given to the Esterel reactive language(Berry, 1999). The behavior of each operator is formally described by a rewriting rule.

Differences between two reactive languages are stated by their semantics. For example, the Esterel(Berry, 1997) synchronous language reacts instantaneously to the absence of a signal, while the SL(Boussinot, 1998) reactive language decides only at the end of the instant that a signal is absent and delays any reaction to this absence to the next instant. In the Esterel language, this semantics generates causality problems : during the execution of one instant, it is possible to decide the absence of some signal, to react to this absence and to continue the execution by emitting the signal, thus breaking the initial absence assumption.

About parallelism and concurrency concepts, it is still difficult to understand how to physically distribute a system made of concurrent sub-systems while preserving the semantics and the hypothesis of the reactive synchronous language (ensure the determinism, predict the behavior, evaluate reaction time, ...). Thus a reactive system, even if it is expressed in a modular way for the sake of expressiveness, is, given the current state of the art, compiled into sequential code.

The semantics given to a reactive language allows us to translate a program into the formal model of finite automaton, either in an explicit way, or in an implicit way using binary decision diagrams. This automaton represents all the possible behaviors of the compiled program, and can be explored with model-checking techniques with the aim to ensure safety and liveliness properties. Notice that for hybrid systems, we are particularly interested in verification involving values, that is why we chose the SMV model-checker.

Most of these reactive programing concepts will be preserved in an approach of hybrid systems design (e.g. the operational semantics leading to compilation into a finite automaton). Some of these concepts, more specific to other domains (e.g. reaction to absence mostly appropriate to hardware design), are useless constraints and should be taken away. Some concepts are totally missing, like a true capability to deal with values.

4. TOWARDS A GLOBAL APPROACH TO HYBRID SYSTEMS

Coarsely, the description of a hybrid system involves a part that evolves continuously with time - modeled by differential equations - and a part that evolves in an event-driven manner - presenting the behavior of a finite automaton -. The evolution of a dynamical system is driven by event occurrence; conversely, various values of the state describing the dynamical system might cause event generation and consequently act on the global behavior of the application.

4.1 *Weakness of a simple co-existence of event-driven and dynamical concepts*

One of our previous applications(Roy and Maïzi, 2000) was about embedding event-driven programs written in the reactive synchronous language Esterel into the simulator of dynamical continuous system Simulink. In this experiment, led through a case-study application in the field of renewable energy systems, the global hybrid systems was expressed under the numerical computational environment matlab.

The weakness of the languages provided by this environment was their absence of semantics. Thus, the global system was relying on a simple co-existence of the various components : a supervising controller and the dynamical physical process. This modeling technique did not allow the analysis of the interactions between these separate parts. Verification was only done on the supervising part, which led to a certain lack of safety. The limitation of this approach was clearly the lack of a general framework that could take into account both parts in an integrated manner.

4.2 *Surrounding a dynamical system behavior by an event-driven conception*

Like purely event-driven system, a hybrid system can be seen as the interconnection of concurrent modules (acting at the same time) communicating together and with the outside through events (signals); each module implementing a part of the global specification (because we want to deal with real-size problems that can be complex, we must be allowed to use modularity and to express systems in a hierarchical way). A module has either a dynamical continuous, or an event-driven behavior. Interconnection between modules is expressed in a natural way in an event-driven manner. The global system is listening to the evolution of the dynamical system through its state variable. Our modeling method offers concurrency and modularity, with the aim to address expressiveness and code reusability.

As soon as we deal with hierarchy, reusability and modularity, we must also mention compositionality. Say that we have built two hybrid systems, that we have validated : they have been specified, implemented, tested, simulated and verified. When designing a new hybrid system, we of course want to re-use those systems as parts of the new one, so that we can take advantage of the work previously done.

The problem is to know whether or not the mere fact of connecting both subsystems together is likely to produce a valid system. Compositionality makes sense for event-driven systems in general, though Esterel language is not compositional. The continuous dynamical systems are compliant with the compositionality constraints, and thus may be, e.g. sequentialized or parallelized. For a hybrid system, we can think of compositionality as an enhancement of compositionality applied to its dynamical aspects : the continuous behaviors must be composed before the event-driven behaviors can be.

We will now detail our example as it is representative of our approach. We are still at the experimental stage, a formal modeling will come later, after the improvements of all the concepts we want to include in our design.

5. OUR EXPERIMENT

Aiming to deal with examples in a realistic manner, our implementation takes advantage of results and tools that appeared these last few years, in the fields of control theory and even-driven systems. For the modeling of dynamical systems to be expressive, complete and meaningful, we take matlab. For the even-driven parts, we benefit of modeling, compilation et simulation tools developed in the domain of the reactive synchronous language Esterel. Finally, we take the model-checker SMV(McMillan, 1999) for its capabilities in the field of verification involving values. Interconnection between both parts is taken into account inside the reactive language. A program will be compiled into a formal model.

Given a continuous dynamical system implemented under the matlab environment, the information actually relevant in the evolution of the hybrid system is its state. So as to be able to access this information in Esterel, we solve step-by-step the equations describing the dynamical

system (from now on, let's call it the *step-by-step simulation*). For that we invoke the dedicated solvers in matlab via Esterel functions. As arguments, we provide matlab with the input of the dynamical system, and in return we get the new state : we retrieve its output for these values to be available from the system and also to provide it to the next simulation step. In order to describe these typed input-output by matlab types, we use Esterel abstract data types.

```
var current_state : MATRIX in
....
   initialize_F();
   every INTEGRATION_STEP do
      actualize_F(previous_state,
                  current_time,
                  control,
                  perturbation)();
      current_state =
         integrate_F(previous_state,
                     current_time,
                     control,
                     perturbation);
      emit SYSTEM_STATE(current_state);
   end every
end var;
```

At most one dynamical system can be enabled at a time in the global hybrid system. Simulating two dynamical systems at the same time and in a separate manner does not make sense : they must be composed to be simulated together. The Esterel language is not able to ensure this restriction. In our example, it is done by programming carefully : each step-by-step simulation is enclosed in an `every INTEGRATION_STEP` statement and we avoid combining step-by-step simulations with a parallel operator.

For implementing this example, we compile Esterel programs into C-code. The integration functions, written in matlab, are translated into calls to external function written in `m`, themselves being translated into C-code by the matlab tool `mcc` that allows how to compile m-code into C-code.

Within this framework, the dynamical system is observed through its trajectory; this trajectory is then "plugged" into an event-driven program i.e. derived into a finite automaton behavior. As far as execution is concerned, firing a transition may have two major types of causes :

- the variable describing the dynamical system status gets outside of a predefined range;
- an external event occurs.

Firing a transition implies the system to switch from one dynamical system to another as described in section 2.1. Our example is about the management of energy flows performed through the regulation process of photovoltaic panels, diesel, user appliances and battery bank. The state of the dynamical system is the state of charge of the battery bank. Whenever the battery bank charge crosses given thresholds, some parts of the global energy system - like photovoltaic panels or diesel or battery bank itself - are connected and disconnected.

SMV is a formal verification tool. This model checker is equipped with a modeling language named synchronous verilog. This language is in the same family as Esterel and thus can be produced from an Esterel program. In our example, the compiled code obtained, which gives every possible behaviors of the program, contains dynamical system step-by-step simulation appearing as function calls. So we can write and verify some properties involving tests on these function return values.

6. CONCLUSION

For validating our pre-integrated design approach of hybrid systems, we developed an example where the continuous parts of the system are fully integrated into an event-driven world. Our implementation relies on dedicated techniques and tools from these two domains. On the continuous side, the matlab computing system is, by its very nature, perfectly suited for developing the dynamical parts. On the event-driven side, the synchronous reactive language Esterel is used for formally modeling the reactive behaviors.

Nevertheless, when used within the hybrid system area, it exhibits several drawbacks. This language, being much oriented towards hardware circuits specification, is obviously not very well suited for handling valuated signals. For instance, the data, as well as the conditional tests where they appear, are not taken into account in the control flow, nor in the finite automata that serves as a basis for verification. This drastically decreases our ability to perform verification of the behaviors involving values, while this is precisely what we are interested in. In much the same way, apart from the primitive types, an elaborated data type can only be an abstract data type whose description must be done in an external *host language*. But within the context of hybrid systems, there is an actual need for dealing with arrays, matrices, records or enumeration, and these constructions need to be built-in.

Moving forward on a similar track, we would like to have specific constructions in the language for interfacing dynamical systems, such as the declaration and invocation of typed dynamical

systems. These constructions are likely to pretty much improve the readability of the interfacing parts of the system.

Lastly, Esterel causality policy is much too heavy for us.

Considering all these remarks, we would like, from now on, to focus our future work on the definition and implementation of a language more specifically dedicated to deal with hybrid systems modeling.

7. REFERENCES

Berry, G. (1997). The esterel primer. Technical report. Ecole des Mines de Paris and INRIA.

Berry, G. (1999). The constructive semantics of pure esterel. Technical report. Ecole des Mines de Paris and INRIA.

Boussinot, F. (1998). *The SL language*. CNET.

F. Baccelli, G. Cohen, G.J. Olsder and J.-P. Quadrat (1992). *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Wiley.

McMillan, K. (1999). Tutorial on smv. Technical report. Cadence Berkeley Labs.

Roy, V. and N. Maïzi (2000). Application of an hybrid esterel, lustre, simulink system : a hybrid energy system. *Automation of Mixed Processes: Hybrid Dynamic Systems: The 4th International Conference*.

Utkin, V. I. (1977). Variable structure systems with sliding mode : A survey. *I.E.E.E. Trans. Automatic Control*.