Using Hybrid Automata to Express Realtime Properties in VSE-II

Andreas Nonnengart Georg Rock Werner Stephan*
Deutsches Forschungszentrum
für Künstliche Intelligenz
Stuhlsatzenhausenweg 3
66123 Saarbrücken

Abstract

In formally analyzing and developing industrial sized systems we are often confronted with the problem of expressing realtime properties. Especially in safety critical applications as, for example, in embedded systems or in control software these properties are crucial for the right functioning of the systems.

There are numerous suitable approaches that allow us to specify realtime properties: MTL, TLA or VSE-SL to name a few, and also there are various tools available to formally develop such systems. But in most cases the approaches are trimmed for their own highly specialized application area. We try to overcome this restriction by exploiting the advantages of a (fairly general) formal development tool like VSE-II and the rather specialized Hybrid Automata by combining these two approaches.

Introduction

In specifying industrial (sized) systems, we are often confronted with the problem of expressing realtime properties of systems. In these applications we realized that these propositions are both hard to find and hard to verify. Those properties could be expressed in many specification languages as, for instance, MTL, TLA or VSE-II. One of the main problems in this area is the complexity of timing constraints that could arise and the variety of situations which could appear in such a system. In writing formulae like

$$\Box(timer \leq 10 \rightarrow \Box(t_0 < now \leq t_1 \rightarrow x = false))$$

which says that whenever $timer$ does not exceed 10 then x remains false throughout the interval $[t_0, t_1]$, we are only able to capture particular aspects of the timing behavior of the system. People specifying such systems often draw diagrams that describe such a particular behavior and they try to find critical situations in this way. It is clear that these diagrams might help to understand the system, but they certainly are not suitable to express the entire temporal behavior of a system. A more adequate way is to exploit timed automata or linear hybrid automata in this context.

We want to use linear hybrid automata as an extension of the formalism used in VSE-II to express (complex) realtime behaviors. It is not our intuition to subsume hybrid automata, we rather want to integrate them in a formal software development process which is fully supported by a tool like VSE-II.

The paper is organized as follows: We start giving short descriptions of VSE-II and hybrid automata. Hereafter known results of the the process of embedding hybrid automata into VSE-II are presented. These results are needed as preliminaries to use hybrid automata as a specification language for realtime properties in VSE-II. The paper finishes with a conclusion and future work to be done in this area.

Formal Systems and Logics

In what follows we present the Verification Support Environment as well as the hybrid automata as they are used in our approach.

VSE-Verification Support Environment

The VSE system is a tool for the formal development of software systems. It consists of: A basic system for editing and type checking specifications and implementations written in the specification language VSE-SL, a facility to display the development structure, a theorem prover for treating the proof obligations arising from development steps as for example refinements, a central database to store all aspects of the development and an automatic management of dependencies between development steps.

Compared to VSE I (Hutter *et al.* 1996a; 1996b), which was based on a simple, non-compositional approach for state based systems, VSE II (Hutter *et al.* 1999) is extended with respect to comprehensive methods in order to deal with *distributed* and *concurrent systems* (Rock, Stephan, and Wolpers 1997) and with respect to an even more efficient and uniform proof support which makes use of implicit structuring of the proof obligations that arise. The basic formalism used in VSE II is close to TLA (Temporal Logic of Actions) (Lamport 1994). A refined *correctness management* allows for an evolutionary software development.

VSE is based on a methodology to use the structure of a given specification (e.g. parameterization, actualization, enrichment, or modules) as a means to distribute the deductive reasoning into local theories (Rock, Stephan, and Wolpers 1999). Each theory is considered as an encapsulated unit, which consists of a local signature and certain reasonable

*nonnengart,rock,stephan@dfki.de

axioms. Relations between different theories, as they are given by the model-theoretic structure of the specification, are represented by different links between theories. Each theory maintains its own set of consequences or lemmata obtained by using local axioms and other formulas included from linked theories.

This method of a structured specification *and* verification is reflected in the central data structure of a *development graph*, the nodes of which correspond to the units mentioned above. It also provides a graphical interface for the system under development.

There are two kinds of VSE specifications: abstract data types and specifications of state transition systems. Abstract data types are defined using full first-order logic. For the specification of state transition systems a specification language close to TLA (Abadi and Lamport 1991; Lamport 1994; Abadi and Lamport 1995) is used. These specifications depend on abstract datatypes for the definition of the states. In addition to the theory of compositional development presented in (Abadi and Lamport 1995), which covers the composition of systems using input and output variables, *shared variables* are supported by the structuring operators in VSE II.

The most important operators provided by VSE II to structure state-based specifications is the **combine** operator. It models the concurrent execution of components. Concurrency is modeled by considering all possible *interleavings* of actions of the combined systems. Basically, a behavior which represents a sequence of states of the specified system is a behavior of the combined system if and only if it is a behavior of every component of the system. In (Abadi and Lamport 1995) environment steps are modeled by stuttering. This technique only works for communication by input-output variables, not in connection with shared variables. A more general approach (Rock, Stephan, and Wolpers 1999; Hutter *et al.* 1999) is to associate a "color" with each component and to mark each step in a behavior with the color of the component that performed the step.

Structuring specifications as described above supports readability and facilitates editing specifications. However, the system exploits structure beyond this purely syntactical level. Components of a combined system can be viewed as systems in their own right where certain parts can be observed from outside while most of the inner structure including the flow of control and local program variables are hidden.

In particular we can prove properties of a combined system in a modular way. This means that we attach local *lemma bases* to components where local proofs are conducted and stored. Exchange of information between lemma bases is on demand. This approach has two main advantages: First, the given structure of the specification is used to reduce the search space in the sense that large parts of the overall system are not visible and second, storing of proofs local to certain lemma bases and making the export and import of information (between lemma bases) explicit supports the *revision* process.

Hybrid Automata

Hybrid Systems (Alur and Dill 1994) are real-time systems that are embedded in analog environments. They contain discrete and continuous components and interact with the physical world through sensors and actuators. Since they typically operate in safety-critical situations, the development of rigorous analysis techniques is of high importance.

A common model for hybrid systems can be found in *hybrid automata*[1]. Briefly, such hybrid automata are finite graphs whose nodes correspond to global states. Such global states represent some sort of general observational situations, as, for instance, "the heater is on" or "the heater is off". During these global states some continuous activity takes place. For example, coming back to the heater from above, depending on the global states, the temperature rises or falls continuously according to a dynamical law until a transition from one node to another one occurs.

These transitions are usually guarded with some constraint formula that is required to hold if the transition is supposed to be taken. Similarly, nodes have some attached constraint formula that describes an invariant for this very node, i.e., some property that has to be true while the system resides within this node. The dynamics of the system's behavior, on the other hand, is given by a description of how the data that interests us changes with time.

Additionally, transitions are annotated with some kind of general assignment that is responsible for the discrete action to be performed by taking the transition.

In order to save space we omit the formal definitions of syntax and semantics of the hybrid automata used in this work. Exact definitions can be found in (Nonnengart, Rock, and Stephan 2001). Furthermore we assume a property specification language PSL (Nonnengart, Rock, and Stephan 2001) for hybrid automata. In its current version PSL merely allows us to write safety formulas. The extension to liveness is mentioned in the future work. PSL was given a linear time semantics which fits best to the semantics of VSE-II specifications.

In what follows we present the general scenario in which we want to use hybrid automata within VSE-II specifications. But before we concentrate on that, we first describe a more special scenario which will later be extended to the general one.

Expressing Realtime Properties using Hybrid Automata

In the previous section we have given short introductions to VSE-II and to (linear) hybrid automata. In the following we present how hybrid automata can be used in the formal software development in VSE-II.

Integrating hybrid automata into VSE-II certainly is a challenging task and one might ask the question what such an integration is good for. Or, in other words, what does it help a formal software engineer if he has such an integrated tool in which he could use both techniques in an interleaved way? It is well known that formal software, we would

[1]Throughout this paper we use the notions *hybrid system* and *hybrid automaton* interchangeably.

rather say formal system development, is a non-trivial task. Working on industrial sized systems using formal methods often results in complicated specifications with complicated proofs of the guarantees the system has to provide. Furthermore in specifying systems in which realtime constraints occur, things do not become easier. We think that some level of complication is in general inevitable. But we could place a tool at the formal software engineers disposal which helps to control this complexity. Until now he could use the VSE-II system which supports the whole formal software development process. But what we want to improve is the support for specifying realtime systems and to give the user a tool at hand where he can choose a simple and maybe even automatic procedure to solve a problem. The first part of the methodology of such a tool is illustrated in Figure 1. Starting point is the specification of the behavior of a real-

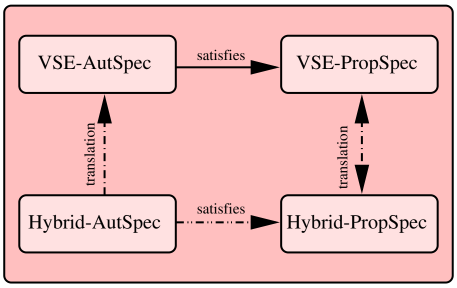

Figure 1: Scenario: VSE - Hybrid Automata

time system using a hybrid automaton `Hybrid-AutSpec` as depicted in Figure 1. Properties that should be satisfied by this automaton are specified in `Hybrid-PropSpec` which is a PSL formula. The idea is to translate (π represents the translation function) the hybrid automaton specification into a VSE-SL specification. From this translated specification, we want to prove that the properties described in `VSE-PropSpec` hold. A part of these properties is created by translating the properties in `Hybrid-PropSpec` into VSE-SL properties. The proof of these properties could be done in the VSE-II tool. The proof could equally be done with a tool supporting hybrid automata.

Some of the advantages of such a method are that we have a more adequate choice of means to specify and verify systems, better support for the specification and verification of realtime systems, and properties and integration of a fully automatic technique in the VSE-II system.

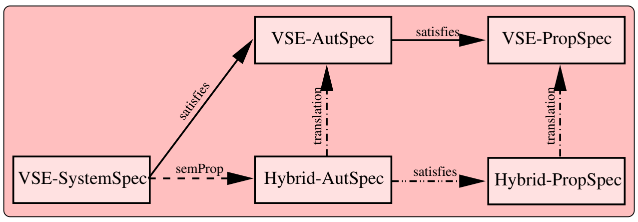

Figure 2: Extended Scenario: VSE- Hybrid Automata

Until now we have discussed all the preliminaries needed to have a closer look at the more general scenario shown in Figure 2. The starting point in the extended scenario is a VSE-II specification (`VSE-SystemSpec`) which describes the behavior of a system in terms of VSE-SL. The realtime properties of this system are expressed by a linear hybrid automaton[2]. The hybrid automaton `Hybrid-AutSpec` has to be translated to a VSE-II specification. After that a satisfies-link between `VSE-SystemSpec` and `VSE-AutSpec` indicates that a proof obligation is generated which has to be proven to show that `VSE-SystemSpec` has the properties expressed by `VSE-AutSpec` that results from the translation procedure π mentioned before. If we are able to prove this, we have established the `semProp` mapping which says that the VSE-II system specification fulfills the hybrid automaton wrt. the translation function π.

```
TLSPEC gasburner
 USING definition
 DATA OUT x, y, t : nat
      OUT state : state_t
 ACTIONS
 A1 ::= (state = leaking AND
         state' = non_leaking AND
         x' = 0 OR
         state = non_leaking AND
         state' = leaking AND
         x >= 30*c AND x' <= c AND
         x' = 0) AND UNCHANGED(y,t)
 A2 ::= state = leaking AND
        state' = non-leaking AND
        NOT x+1 <= c AND
        x' = 0 AND
        UNCHANGED(y,t)
 A3 ::= state = leaking AND
        state' = leaking AND
        x+1 <= c AND y' = y+1 AND
        x' = x+1 AND t' = t+1 OR
        state = non_leaking AND
        state' = non-leaking AND
        x' = x+1 AND y' = y+1 AND
        UNCHANGED(t)
 SPEC INITIAL x=0 AND y=0 AND
      t=0 AND state=leaking
      TRANSITIONS [A1, A2, A3]{x,y,t,state}
 SATISFIES gasprop
TLSPECEND
```

Figure 3: Gasburner as VSE-II specification

The main advantage of our approach to be stressed in this paper is that we are able to combine formal specifications of technical scenarios with the abstract global view of hybrid systems which in this context are considered as comprehensive descriptions of the desired temporal behavior. The technical scenario contains the abstract specification of the actual software system which can be refined to a running program as one of its components.

[2]It is clear that the specification of this automaton has to obey some restrictions wrt. to the system specification and to the expressivity of hybrid automata.

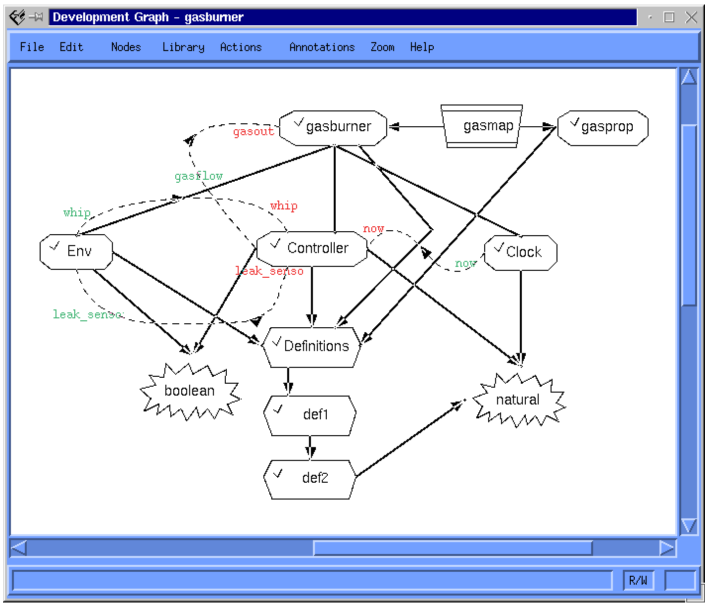

Figure 4: VSE-II Development Graph

As an example case we consider a scenario that consists of the *controller*, the *environment*, and a *clock* (see Figure 4). All three are modeled by concurrent components in VSE. The combined system then is the *gasburner* (scenario).

The specification of the *controller* which is made up of a single action is shown in Figure 5. In one execution cycle the controller reads the current input values to corresponding internal variables and sets output values depending on the previously stored inputs. This can be thought of as a simplified abstract model for the cyclic computation of programmable controllers.

In our simple scenario, which is still far from being a real system, we have as inputs a *leak sensor* and a *whip* for switching the gasburner on and off from outside. The only output is an actuator that opens or blocks the *flow of gas* . Apart from the variables needed to store the input values there is an *internal state* which can be *on* or *off* and an additional *timer* variable. The *environment* sets the input lines read by the controller having access to the current time provided by the clock and the value of the gas actuator. There are several restrictions for the behavior of the environment which model assumptions about the physical world. So for example, we can have $leak = true$ only if $gasflow = open$. The *clock* component provides the global time of the scenario by a variable now. The value of this variable is increased by one when the clock ticks. Since not all intermediate states between two ticks can be regarded as representing physical situations of the scenario an external observer should notice a change of (the value of) a visible variable only if the clock ticks. In the VSE model of the storm surge barrier presented in (Rock, Stephan, and Brodski 2000) there was an explicit update of the visible variables upon each tick of the clock. Of course the clock component needs access to all data exchanged by the components.

```
TLSPEC Controller
 USING natural; boolean; Definitions
 DATA INTERNAL leak_sensor_i : bool
      INTERNAL now_i,timer : nat
      INTERNAL whip_i,cstate : OnOff_t
      OUT gasflow : gas_t
      IN leak_sensor : bool
      IN now : nat
      IN whip : OnOff_t
 ACTIONS
 A1::=
   whip_i' = whip AND now_i' = now AND
   leak_sensor_i' = leak_sensor AND
   (cstate = on AND whip_i = off AND
    leak_sensor_i=F AND cstate'=off AND
    gasflow' = blocked AND
    UNCHANGED(timer,now_i,
             leak_sensor_i,whip_i) OR
    cstate = on AND whip_i = off AND
    leak_sensor_i=T AND cstate'=off AND
    gasflow'=blocked AND timer'=now AND
    UNCHANGED(now_i,leak_sensor_i,
              whip_i) OR
    cstate = on AND whip_i = on AND
    leak_sensor_i = F AND
    UNCHANGED(cstate, timer, gasflow,
            now_i,leak_sensor_i,whip_i)
    OR cstate = on AND whip_i = on AND
    leak_sensor_i=T AND cstate'=off AND
    gasflow'=blocked AND timer'=now AND
    UNCHANGED(now_i,leak_sensor_i,
              whip_i) OR
    cstate = off AND whip_i = on AND
    now_i > timer + (30 * c) AND
    cstate'=on AND gasflow'=open AND
    UNCHANGED(timer,now_i,
             leak_sensor_i,whip_i) OR
    cstate = off AND whip_i = on AND
    NOT now_i > timer + (30 * c) AND
    UNCHANGED(cstate,gasflow timer,
    now_i,leak_sensor_i,whip_i) OR
    cstate = off AND whip_i = off AND
    UNCHANGED(cstate,gasflow timer,
         now_i,leak_sensor_i,whip_i))
 SPEC INITIAL gasflow=0 AND cstate=on
             AND timer=0 AND
             leak_sensor_i=T AND
             now_i=0 AND whip_i=on
      TRANSITIONS [A1]
              {gasflow,timer,cstate}
TLSPECEND
```

Figure 5: Controller as Temporal Specification

Our aim is to interpret this technical scenario as a potential model of the hybrid automaton shown in Figure 6 which is translated to the VSE-II specification shown in Figure 3. This is done by recomputing the values mentioned in the definition of the automaton upon each tick of the clock taking the data provided by the scenario as inputs. Among others the clock would contain as (disjunctively related) subactions $state' = leaking \wedge leaking = true$, $state' = nonleaking \wedge leaking = false$, and $y' = now$. In the automaton shown in Figure 6 t is intended to count the time the gasburner is leaking. The interpretation of this in our scenario is given by the subactions $t' = t + 1 \wedge leaking = true$ and $t' = 0 \wedge leaking = false$.

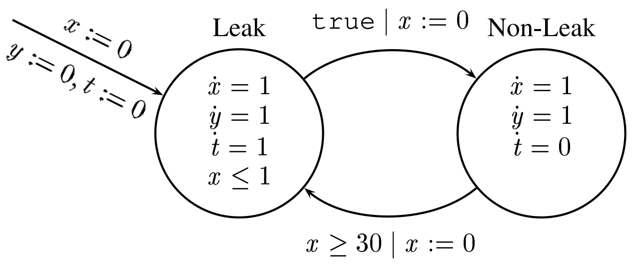

Figure 6: Gasburner as Hybrid Automaton

To model time constraints we need a scheduling strategy for the three components of the scenario. Technically scheduling is realized by activating enabling or disabling components[3]. Let us assume that the controller is a synchronous device and the (fixed) time for one cycle is n. Then after being enabled for one step the controller component has to be blocked for at least n ticks of the clock. It has, however, to be unblocked before $2n$ ticks. In order to satisfy the timing behavior given by the automaton we need constraints for n and c. The general technique which could be considered as a refinement of fairness has been described in (Rock, Stephan, and Brodski 2000).

Conclusion and Future Work

We have presented a methodology to use hybrid automata in the VSE-II formal system development. Hybrid automata are suited to express realtime behavior and VSE-II is suited to formally develop industrial sized systems. Exploiting the advantages of both formalisms we get an adequate means to specify realtime properties of complex systems. We have illustrated the use of this methodology with the simple gasburner example.

Our future work in this field is relatively widespread. In integrating hybrid automata into VSE-II we have developed a translation function π which performs an exact discretization of continuous behaviors (Nonnengart, Rock, and Stephan 2001). This is done for safety properties and has to be extended to liveness properties.

A further point is the exploitation of component based hybrid automata specifications which are synchronized by so called synchronization labels. It is an interesting question how this structure could be translated to VSE-II specifications and whether an assumption-commitment specification style has to be used in the VSE-II specification.

At the end of our work there stands an integrated tool consisting mainly of the VSE-II system as it is now, but with an additional component that allows us to utilize hybrid automata directly or as a specification utility. With this we will have a more adequate means to specify and verify systems with respect to realtime related developments.

[3]For sake of readability this part has been omitted from the specification of the controller.

References

Abadi, M., and Lamport, L. 1991. The existence of refinement mappings. *TCS* 82(2):253–284.

Abadi, M., and Lamport, L. 1995. Conjoining specifications. *TOPLAS* 17(3):507–534.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theoretical Computer Science* 126:183–235.

Hutter, D.; Langenstein, B.; Sengler, C.; Siekmann, J. H.; Stephan, W.; and Wolpers, A. 1996a. Deduction in the Verification Support Environment (VSE). In Gaudel, M.-C., and Woodcock, J., eds., *Proceedings Formal Methods Europe 1996: Industrial Benefits and Advances in Formal Methods*. SPRINGER.

Hutter, D.; Langenstein, B.; Sengler, C.; Siekmann, J. H.; Stephan, W.; and Wolpers, A. 1996b. Verification support environment (VSE). *High Integrity Systems* 1(6):523–530.

Hutter, D.; Mantel, H.; Rock, G.; Stephan, W.; Wolpers, A.; Balser, M.; Reif, W.; Schellhorn, G.; and Stenzel, K. 1999. VSE: Controlling the Complexity in Formal Software Development. In Hutter, D.; Stephan, W.; Traverso, P.; and Ullmann, M., eds., *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*. Boppard, Germany: Springer-Verlag, LNCS 1641.

Lamport, L. 1994. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems* 16(3).

Nonnengart, A.; Rock, G.; and Stephan, W. 2001. Hybrid Systems in VSE-II. In Schellhorn, G., and Reif, W., eds., *Journal of Universal Computer Science (J.UCS)*. Springer Verlag, Heidelberg. to appear.

Rock, G.; Stephan, W.; and Brodski, M. 2000. Modelling, Specification and Verification of an Emergency Closing System. In Etheredge, J., and Manaris, B., eds., *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, 356–360. AAAI Press.

Rock, G.; Stephan, W.; and Wolpers, A. 1997. Tool support for the compositional development of distributed systems. In *Tagungsband 7. GI/ITG-Fachgespräch Formale Beschreibungstechniken für verteilte Systeme*, number 315 in GMD Studien. GMD.

Rock, G.; Stephan, W.; and Wolpers, A. 1999. Modular reasoning about structured TLA specifications. In Berghammer, R., and Lakhnech, Y., eds., *Tool Support for System Specification, Development and Verification*, Advances in Computing Science, 217–229. Springer, Wien-NewYork.