Combining Theorem Proving and Continuous Models in Synchronous Design

Simin Nadjm-Tehrani[1] and Ove Åkerlund[2]

[1] simin@ida.liu.se, Dept. of Computer and Information Science, Linköping University
S-581 83 Linköping, Sweden
[2] ove.akerlund@saab.se, Saab AB, S-581 88 Linköping, Sweden

Abstract. Support for system specification in terms of modelling and simulation environments has become a common practice in safety-critical applications. Also, a current trend is the automatic code-generation, and integration with formal methods tools in terms of translators from a high level design – often using common intermediate languages.
What is missing from current formal methods tools is a well-founded integration of models for different parts of a system, being software/hardware or control-intensive/data-intensive. By hardware we mean here the full range of domains in engineering systems including mechanics, hydraulics, electronics. Thus, there is a methodological gap for proving system properties from semantically well-defined descriptions of the parts.
We report on the progress achieved with the European SYRF project with regard to verification of integrated analog/discrete systems. The project pursues the development of new theories, application to case studies, and tool development in parallel. We use a ventilation control system, a case study provided by Saab Aerospace, to illustrate the work in progress on how hardware and software models used by engineers can be derived, composed and analysed for satisfaction of safety and timeliness properties.

Keywords: control system, synchronous languages, theorem proving, hybrid system, proof methodology

1 Introduction

Many applications of formal methods in system development are in the requirements specification phase – often formalising a subset of requirements corresponding to functional behaviour of the system [9, 6]. In embedded systems, these requirements commonly refer to the component which is under design – typically the controller for some physical devices (realised either as software or electronics). However, there is a class of properties arising as a result of interaction between the controller and the controlled environment, the verification of which requires an explicit model of the environment. This paper addresses

verification methodologies for such types of requirements in the context of synchronous languages.

A growingly popular approach to controller design or programming uses the family of synchronous languages (Lustre, Esterel, Signal and statecharts) [7, 8]. One reason for choosing such languages is the support provided in the development environments: the controller can be analysed to eliminate causal inconsistencies, and to detect nondeterminism in the reactive software. The clock calculii in Lustre and Signal, as well as constructive semantics in Esterel can be seen as verification support provided directly by the compiler (comparable to several properties verified by model checking in [5]). Most of the works reported within this community, however, apply verification techniques to check the controller *on its own*.

Modelling the controlled environment is common in control engineering. However, the analysis tools within this field primarily provide support for continuous system simulation, and are less adequate for proving properties of programs with discrete mode changes and (or) complex non-linear dynamics in the plant.

Within the Esprit project SYRF (on SYnchronous Reactive Formalisms), we present an approach whereby modelling tools used for analysis of analog systems can be used to substantiate the properties of the environment when formally verifying a closed loop system. We use the continuous model of the environment in two different settings. In the first approach, compositional verification is performed across different modelling platforms [14]. A required property is split into a number of conjuncts (proof obligations). Some of these are discharged by proofs in the discrete platform, using the controller properties. Others are verified in the environment model by simulation and extreme case analysis. Certain properties are refined in several steps before they are reduced to dischargable components.

In the second approach we model (aspects of) the continuous subsystem in the same (discrete) proof environment as the controller. Here, the restrictions in the physical model provide a sufficient condition: The proof of the property in the closed loop model holds provided that the restrictions leading to the discretised model holds.

A case study provided by Saab Aerospace is used to illustrate the alternative approaches, the properties for which they are appropriate, and some verification results obtained. However, some comparative studies are still in progress, and will be conclusively presented in the final report of the project.

2 The air control case study

The case study consists of a climatic chamber. A control system regulates and monitors the flow and the temperature of air which is circulating in the chamber. Originally, it was developed as a demo system which demonstrates the kind of problems appearing in developing realistic subsystems such as the ventilation system in the JAS 39 Gripen aircraft. It was presented to the project partners in

terms of a 4 page textual specification and an implemented code for a controller in hierarchical block diagrams with state machines at the lowest level.

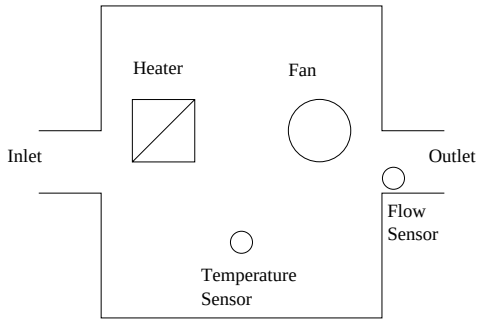

Fig. 1. The hardware components of the air control system.

The chamber is to be ventilated through the inlet and outlet and has a given volume. It has two sensors for measuring the internal air temperature and the air flow. Figure 1 presents the component model of the chamber, while Figure 2 shows the interface between the system and the operator. The external interface primarily consists of an on-off button, two analog knobs for setting the values for required temperature and flow (reference values), as well as warning signals in terms of a light and a sound. It also includes lights for showing some of its internal modes of operation.

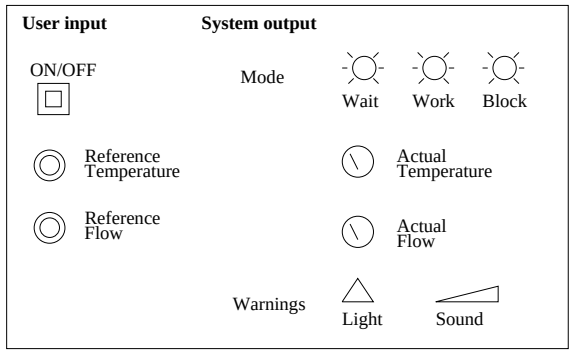

Fig. 2. The external interface to the system.

The controller has three modes while it is on. It has an initialising "wait" mode in which the heater and the fan are used to bring the chamber temperature and flow within a given scope. It also has two active modes in which more accurate regulation is achieved. One is the "solution" mode in which the actual

temperature and flow values are brought to levels close to the reference values. The other, the "work" mode in which the actual values are maintained in the required region (within Δ of the reference values). The final mode, denoted as the "block" mode, is devoted to abnormal situations and it is intended as a shut-down mode. It is brought about when the earlier sound and light warnings have not led to changes in the reference values by the operator, or when the actual values fall outside the allowed scope despite manual intervention (for example due to unforeseen changes in unmodelled inputs, e.g. the incoming air temperature).

2.1 Requirements specifications

The textual description mentioned above has a prescriptive nature. It describes how a controller should be implemented, giving some details about what should happen in each mode. To focus the formal verification work we had to deduce the overall goals of the control system: those requirements which are to be enforced by the suggested design.

The result of this study has been identification of the following global requirements.

- Keeping the reference values constant,
  - the work light will be lit within a time bound from the start of the system, and
  - the system will be stable in the work mode.
- Chamber temperature never exceeds a given (hazardous) limit.
- Whenever the reference values are (re)set, the system will (re)stablise within a time bound or warnings are issued.

Note that these are not properties of the controller on its own. Note also that our formulations are intended to fit in a framework where different proof techniques are applied where they suit best. Although "being stable in the work mode" can be seen as a safety property (the conditions for leaving the mode will not be true), it is most expedient to use control theory methods for proving this property. This is due to the fact that not *all* inputs to the system are kept constant (see the result of the physical modelling step). Hence, it is formulated as a stability property.

Another aspect to point out is on the second (safety) property. Here we look beyond the functional demand on the system to monitor and warn when the temperature falls outside given intervals. We rather attempt to see what is the goal of devising such intervals and mode changes and envisage as a (mode-independent) goal of the system that the air is never heated to a hazardous level (even in the block mode and *after* warnings are issued).

3 Model of the controller

The controller has been modelled in several synchronous languages both in the data flow style (Lustre), and the control flow style (Esterel, statecharts). It

represents the typical case where it is most naturally described in a combination of these paradigms. Thus, mode automata [12] and the synchronie workbench [1] use this as a demonstrator system. Also, a multi-formalism representation of the example used for distributed code generation can be found in [3].

Having models which reflect the nature of the computations naturally, surely avoids some development errors. Moreover, once the models are analysed with respect to the required properties they can be automatically translated to intermediate and lower layer programming languages. For example from mode-automata to Lustre, to DC, and to C (see work package 2 in the project [17]). Note that code generation is also available in tools which support analysis of continuous systems and analog (periodic) controllers (e.g. Matlab and MatrixX [10]). However, these are not targeted for cases with complex software with hierarchical structure and do not support formal verification.

It is also essential to obtain integration with analysis tools if the detailed design is to be formally verified prior to code generation. This is much more obvious where the controller has a hierarchical description, discrete mode changes, and complex control structures. Here, the work in the project is still in preliminary stages. Prototype translators from Lustre to PVS [16], and Lustre to the first order theorem prover NP-Tools by Prover technology have been developed (see work package 3.4 in [17]). However, the applications are still in progress.

Here we report on one such translator used in the case study: the prototype developed by Prover technology which translates a subset of the Statemate languages (with a synchronous interpretation) to NP-Tools with integer arithmetic [4]. The model of the controller in statecharts is too large for being presented here. However, the size of the translated NP-Tools model provides a feel for the size. The insect-like macro resulting from the translation to NP-Tools has 96 input variables and 88 output variables (seen as a circuit).

3.1 Lessons learnt

Our experience with the modelling activities for climatic chamber controller can be summarised as follows. The NP-Tools [18] environment should obviously be seen as an analysis environment, not a primary modelling environment. The description of the controller at the circuit level loses much of the inherent structure and does not provide an overview when compared with the statechart model. On the other hand, using statecharts alone was not ideal for description of such a controller either. The model we developed prior to translation to NP-Tools used only a subset of the Statemate (statechart) notation. In particular, activity charts could not be used. Thus, all (continuous) control activities which are ideally described in a language like Lustre give rise to several self-loops within every active regulation mode, each loop having its own enabling condition.

The result of the translation from statecharts to NP-Tools was a macro with all the inner logic hidden. Each dynamic variable was modelled as an in-pin representing the value before each step, and an out-pin for the value after the step (additional pins for initial values are also provided). During the verification step counter-models presented by the theorem prover showed errors in the design

model. However, after every modification to the design (in the statechart model), one needed to recompile to the NPTool format, which soon became impractical.

As a result of the childhood problems with the translators, we have so far attempted all our closed loop verifications on models directly developed in NP-Tools and a physical environment model. When modelling in NP-Tools we have used a similar style to modelling (variable naming conventions for values before and after a step, etc), as if the model was the result of translation from the statechart model.

The experience here shows, however, that much of the value in high level modelling is lost. To show, for example, that the control system is only in one mode at any time produced a number of counter examples and several modifications to the model. This is trivially achieved by competent compilers (e.g. the Esterel compiler based on constructive semantics [2]).

We are currently experimenting with the Lucifer tool which is a similar translator from Lustre to NP-Tools (see SYRF deliverable 2.2 [17]). Here, translation provides an improvement. The hierarchical structure of the Lustre program, not so visible in the textual language, becomes more visible in the NP-Tools version. This is due to preservation of the structure at the Lustre "node" level (one NP-Tools macro for each Lustre node).

4 Models of the physical environment

The physical model developed for the climatic chamber case study and the underlying assumptions were detailed in [14]. In the simplest form, the continuous model for the example, as derived from engineering models, has one differential equation describing changes in the chamber temperature as a function of three inputs: the incoming air temperature, the applied voltage, and the air flow in the chamber.

An initial hybrid model for this part (under the given assumptions) is seemingly simple: consisting of one discrete mode and one equation. The differential equation, in which u_i are inputs, x is the only state variable, and a, b and c are constants, has the following form:

$$\dot{x} = au_1x + bu_2 + cu_1u_3$$

Here, u_1 denotes the air flow $[m^2/s]$, u_2 is the square of the controller-applied voltage $[V]$, and u_3 is the temperature for the incoming air $[K]$. x denotes the chamber temperature which is prescribed to be within allowed ranges in different modes by the requirements/design document. Namely, the document refers to the chamber temperature being "within Δ of the reference temperature", or being " within 2Δ the reference temperature" as part of the transition condition between various modes.

4.1 Transformations on the model

Ideally we would like to combine this model and the synchronous controllers described above, and perform analysis on the closed loop system. However, pro-

totypical analysis environments in which hybrid models can be analysed are much more restrictive. Note that despite simplifying assumptions this model is still non-linear, and in particular the evolutions in state are not linear in time. We therefore propose a number of transformations on the model which makes some specific instances of it analysable. Two obvious "specialisations" are transformation to hybrid automata (HA) and transformation to a discrete time model.

Thus, we look at certain restrictions to the model which yield a "simpler" representation. Though simplicity might mean a larger number of discrete modes with simpler dynamics in each mode.

Another reason for looking at these restrictions is that the environment model above is an open system. One of the "simpler" models, hybrid automata, requires us to give invariances over every mode and differential equations describing each variable of the system. The distinction between state and input is thus removed, and the model is expected to incorporate full information both about control signals (here the voltage), and the disturbances (here the incoming air temperature).

On the other hand, we wish to keep a modular version of the environment (although simpler). We would like to plug and play with different control programs and verify each property in that context. Thus, there is a conflict between making the model simpler (e.g. turning it into HA) and keeping it modular.

We therefore propose a number of restrictions which can be applied with as little impact on modularity as possible. In particular, we distinguish between restricting:

- unmodelled inputs, and
- modelled inputs.

With unmodelled inputs we mean those which are completely outside our control. In the context of the case study the incoming air temperature is such an input. Since we do not have any information on how they can vary, restriction to a class, in any case proves something about the closed loop system when inputs are in that class. For these inputs we assume piecewise constant signals with a finite range of values.

For modelled inputs, either the input is described in detail as the state of another continuous state system, or the input is a control signal generated by a control program. In the former case, a parallel composition of the hybrid transition system eliminates those variables as inputs and makes them state variables. In the latter case – for control signals – we again restrict the signal to a class without making the controller behaviour fixed. In particular, control signals issued from a synchronous controller, depending on being periodic or not, lead to different abstractions of the physical model.

(a) piecewise constant control signal with changes allowed at equidistant points in time, lead to the discrete-time abstraction of the model as difference equations.

(b) piecewise constant control signals which set the rate of change of a continuous variable (e.g. increase, decrease, steady), lead to piecewise constant slopes incorporated in a hybrid automaton model.

We attempt both approximations in the project case study (see section 5 below). As far as other continuous (non-control) inputs are concerned, as a first approximation it is reasonable to assume that they are constant. This is standard practice in control engineering, and again, gives valid results for those system trajectories brought about by the constant input.

In the climatic chamber, as a first approximation we assume that the flow (u_1)is constant at all times. We further assume that the incoming air temperature (u_3) is piecewise constant with a finite range of values.

Thus the model of the system can be transformed to a hybrid transition system (HTS) [15] with one mode for every possible value of u_3. This analysis gives us Figure 3 as the first approximation.

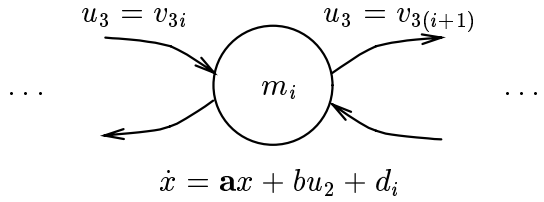

Fig. 3. Assuming that domain of $u_3 = \{v_{31}, \ldots, v_{3n}\}$, $\mathbf{a} = au_1$, and $d_i = cu_1v_{3i}$

4.2 The hybrid automaton model

One restriction to the incoming control signal assumes the controller to have three modes of operation with regard to the control signal: increasing, decreasing and keeping constant. In this section we take the model of Figure 3 and restrict u_2 to be of this type. This assumption leads to a model of the chamber whereby every mode in Figure 3 will be replaced by three modes as displayed in Figure 4. The figure shows the obtained hybrid automaton fragment, where the conditions for incoming and outgoing transitions from the fragment are left out.

It should be clear that by specifically stating the rate of change for u_2 this variable can no longer be considered as an input variable in the original transition system. In order to utilise the added knowledge for simplifying the equation for x, we need to relate rate of change of u_2 with the changes in x. Thus, we need to explicitly represent a clock which measures how long the system has resided in each mode since the last time it was entered. We use the clock t for this purpose. This variable has to be reset to zero every time a mode is entered. Furthermore, we need to establish the condition for leaving a mode and entering a new one,

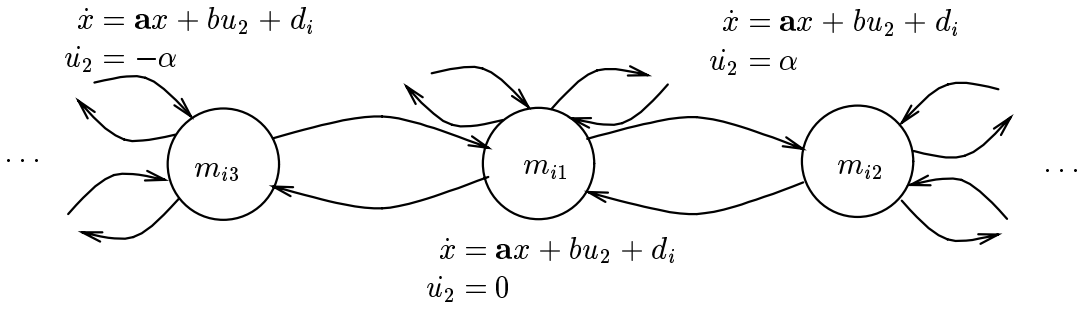

Fig. 4. An HTS model with u_2 as a state variable: assuming that it may stay constant, or increase/decrease at a constant rate.

which obviously arises due to actions of the controller. In HA, this is achieved by adding synchronisation labels corresponding to controller actions. The HA in Figure 5 allows arbitrary changes of slope within the range $\{-\alpha, 0, \alpha\}$ by the controller. Note that the value of x is now dependent on the (apriori unknown) value of u_2 on entry to the mode. This value on entry is captured by adding a piece-wise constant variable (g_i) and an assignment at each mode change.

It can be observed that the obtained hybrid automaton still is not analysable algorithmically. That is, it is not yet a *linear* hybrid automaton (x does not vary with constant slope). To make this model analysable using the existing (hybrid automata) verification tools, we need to add bounds on the evolution of x (otherwise the value of x will increase or decrease infinitely as time goes by in each mode). Adding these bounds is possible once the plant model is composed with a particular controller – a controller which has output signals of the type assumed in this plant model, i.e. an on-off controller with three values for u_2. Since the Saab program is not of this type this track will not be continued any further.

4.3 The discrete time model

Considering constant flow and piecewise constant incoming air temperature as in previous case, but a different restriction for the control signal we obtain a different approximation in this subsection.

Here, we assume that the heater is controlled by a synchronous program. Moreover we assume the incoming control signal (voltage) and its square u_2 to change only at equidistant points in time. After this assumption, one can rewrite the differential equations into a discrete-time form by making the sampling interval T a parameter of the model. Thus, every differential equation in Figure 3 may be replaced by the difference equation:

$$x((k+1)T) = x(kT)e^{\mathbf{a}T} + b/\mathbf{a}(e^{\mathbf{a}T} - 1)\ u_2(kT) + d_i/\mathbf{a}(e^{\mathbf{a}T} - 1)$$

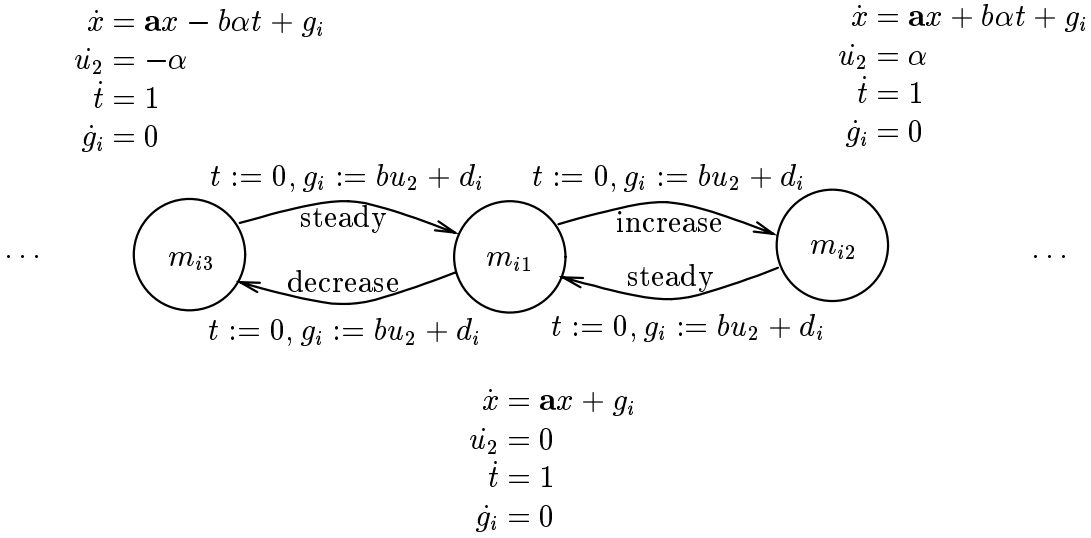

Fig. 5. Fragment of a hybrid automaton model with the same assumptions as in Figure 4 – the clock t and the piece-wise constant function $g_i = bu_2 + d_i$ have been added to relate the changes in x to u_2.

That is, the $(k+1)$th value of x is defined in terms of the kth value of x and the kth value of u_2 (which is assumed constant during the interval $[kT, (k+1)T]$). This reduces the chamber model to a mode-automaton which is a hierachical model compilable to a Lustre program [12]. The syntax and semantics of mode-automata can also be found in the SYRF deliverable 2.1 [17].

4.4 Lessons learnt

In the last two subsections we have seen how treatments of the control signal in two different ways results in two different "simplified" models, each useful in the context of some verification environment (see section 5).

Alhough it might seem that these guidelines are ad hoc, they rest on underlying general principles which justifies them in the context of verification. For example, to restrict the control signal in the above two ways is definitely superior to treatment of such a signal in a way similar to unmodelled inputs. Consider for example the case that the input u_2 (representing the square of the issued voltage) is piecewise constant with a finite range (with no further restrictions).

This leads to a new model, starting from the HTS in Figure 3 and repeating the same step earlier performed for u_3. That is, the voltage signalis assumed to have a finite range of values leading to the finite range $\{v_{21}, \ldots, v_{2p}\}$ for u_2. Replacing every mode of the HTS in Figure 3 with p modes, we get a totally connected HTS of the form shown in Figure 6.

Note that "simplifying" with the same treatment for two different types of input variables gives different results. In the case of a physical variable (the

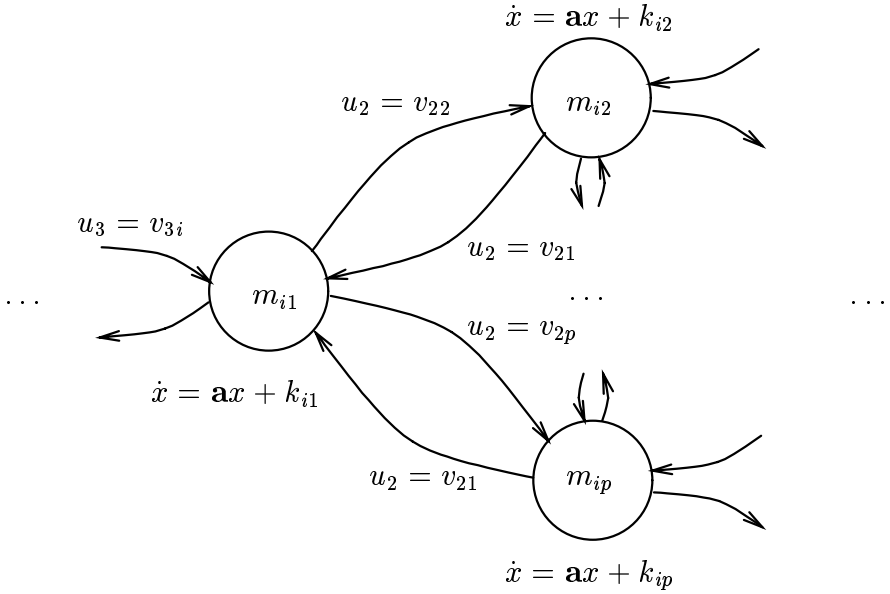

Fig. 6. The HTS obtained with piecewise constant restriction on the control variable u_2 – where $k_{ij} = bv_{2j} + d_i$.

incoming temperature), it is reasonable to assume that values $v_{31}, \ldots, v_{3n}$ can be taken by u_3 in that order. In the case of the control signal u_2 we should assume that the variable may be set to any of the values $v_{21}, \ldots, v_{2p}$ in any order. We simply have no continuity assumptions on a discrete signal. Here, simplification of the continuous dynamics in a mode comes at a much higher price in terms of the increase in the number of (discrete) modes.

In those cases where the nature of the controller is intentionally left open (e.g. not restricted to be periodic) this might be a suitable abstraction. However, it is unnecessarily complex if we already intend to test a particular controller with specific characteristics (on/off controller in the HA case, and a sampled program in the case of the discrete time model).

5 Verification techniques

In the project we have experimented with two different approaches to verifcation. The first one is compositional: a given requirement is decomposed into several conjuncts (prrof obligations). Different subsystem models (represented in different modelling environments) are used to verify that different proof obligations hold. The second approach, referred to as one-shot verification, models the physical system in the same proof environment and at the same abstraction level as the controller.

5.1 Compositional verification

Our approach combines formal and informal reasoning as well as continuous analysis. In this approach we combine proofs in the NP-Tools theorem prover and simulations in the SystemBuild environment of the MatrixX tool [10].

First, we attempt to find sufficient conditions which facilitate proving a property using our knowledge of the system. These auxiliary properties may be of the following kinds:

- an assumption which we discharge informally
- a property of the controller or the environment which we formally prove locally
- another property arising as an interaction of the two, which we further refine by finding further sufficient conditions

Then the system satisfies the top requirement under the informally discharged assumptions.

Consider the second property which is a safety property. The only actuator in the system causing hazards is the heater which must be shown to heat the air to desired levels but *n*ot to hazardous levels. Let $\mathbf{R_2}$ express this property.

$\mathbf{R_2}$: The chamber temperature x never exceeds a limit T_H

The aim is to find (strong enough) properties $\mathbf{R_{2i}}$ such that $\bigwedge R_{2i}$ is sufficient for proving $\mathbf{R_2}$. We start with the following conditions:

$\mathbf{R_{20}}$: The chamber temperature is equal to the incoming temperature u_3 at start time
$\mathbf{R_{21}}$: The reference temperature T_{Ref} can never exceed $T_{Ref_{max}}$, and $T_{Ref_{max}} + 2\Delta < T_H$
$\mathbf{R_{22}}$: Whenever the system is in wait-, solution-, or work-mode, we have $x < T_H$
$\mathbf{R_{23}}$: The system is never in block-mode while $x > T_H$

These properties can be discharged informally or proved within the NP-Tools model except for $\mathbf{R_{23}}$ which we continue to refine:

$\mathbf{R_{231}}$: $x = T_{Ref} + 2\Delta < T_H$ when entering the block-mode
$\mathbf{R_{232}}$: the applied voltage $u = 0$ throughout the stay in block-mode
$\mathbf{R_{233}}$: The system leaves the block-mode after *tblock* seconds, and enters the off-mode
$\mathbf{R_{234}}$: The temperature x does not increase while the system is in the block mode

This is sufficient for proving the safety property provided that
$\mathbf{R_{231}} \wedge \mathbf{R_{232}} \wedge \mathbf{R_{233}} \wedge \mathbf{R_{234}} \rightarrow \mathbf{R_{23}}$.
Properties $\mathbf{R_{231}}$ to $\mathbf{R_{233}}$ are easily proved using the NP-Tools model of the controller. For the proof of $\mathbf{R_{234}}$ we use continuous reasoning based on the simulation models.

5.2 One-shot verification

Here we describe the approach whereby some aspects of the environment model are directly stated in the same verification environment as the controller is.

Consider now the first requirement. The stability component of this requirement can best be verified using control theory and exact knowledge of the control algorithm in the work mode. Here, we concentrate on the first component, denoting it by $\mathbf{R_1}$.

> $\mathbf{R_1}$: Keeping the reference values constant, the work light will be lit within t_1 from the start of the system

First, we provide sufficient conditions for $\mathbf{R_1}$ to hold in the design model:

> $\mathbf{R_{11}}$: The system starts in the wait mode with the chamber temperature equal to u_3
> $\mathbf{R_{12}}$: While T_{Ref} is constant, the only successor to the wait mode is the solution mode
> Given input restrictions $\mathbf{R_{10}}$,
>> $\mathbf{R_{13}}$: The system leaves the wait mode within *wait_time* from the start of the system
>> $\mathbf{R_{14}}$: the system leaves the solution mode within *solution_time* from entering the mode
>
> $\mathbf{R_{15}}$: While T_{Ref} is constant, the only successor to the solution mode is the work mode, and the work light is turned on whenever work mode is entered
> $\mathbf{R_{16}}$: $wait\_time + solution\_time \leq t_1$

We initially claim that

$$\mathbf{R_{11}} \wedge \mathbf{R_{12}} \wedge \mathbf{R_{13}} \wedge \mathbf{R_{14}} \wedge \mathbf{R_{15}} \wedge \mathbf{R_{16}} \rightarrow \mathbf{R_1}$$

At a later stage we may drop $\mathbf{R_{11}}$ and replace it with the assumption that the initial chamber temperature is different from u_3. But to begin with, we make the restrictions in $\mathbf{R_{10}}$ more explicit, and show that

$$\mathbf{R_{10}} \rightarrow \mathbf{R_{13}} \wedge \mathbf{R_{14}}$$

Here, we have several paths to take, but the choice is guided by the verification techniques we intend to utilise. For example, the following restrictions justify the adoption of a discrete-time model of the environment in a mode-automaton [12] with n discrete modes. Each mode is then governed by a difference equation derived from the continuous model (see section 4.3) in the standard manner.

> $\mathbf{R_{101}}$: u_1 stays constant at Q $[m^3/s]$
> $\mathbf{R_{102}}$: u_2 may vary every t_{sample} seconds
> $\mathbf{R_{103}}$: u_3 is piecewise constant taking the values $\{v_1, \ldots, v_n\}$

Note that using mode-automata [12], changes in state variables in each mode are defined in terms of a Lustre program. Each state variable is thus defined by an equation relating the state variables at the previous (clock) step and the current input.

Adopting the restrictions above, the verification method would be as follows: using a scheme for compilation from mode-automata to Lustre we obtain a model of the environment in Lustre which can be composed with a controller in Lustre, and further compiled to NP-Tools. In NP-Tools it is possible (but tedious) to show that the number of steps leading to the work light coming on is $\leq N$ for some N (this proves $\mathbf{R_1}$ for a given t_{sample} provided that $t_1 \geq N t_{sample}$).

The tool Lucifer which translates Lustre programs to NP-Tools models makes these proofs easier. It facilitates inductive proofs with a base larger than 1. That is, it is possible to compose n copies of the transition relation for the system, and show the initial condition holding in the first n steps, followed by the inductive step. This is a track we are currently exploring in the project.

Note that this is one reason for not choosing a "too short" sampling interval [14]. As well as other disadvantages associated with oversampling, a large N makes the proof more difficult. Our approach is based on proving the bounded response property for as small N as feasible.

6 Related works

The work we have reported is at a too early stage for making definitive remarks about feasibility of combining "push-botton" theorem provers and simulation environments. More work is also needed to compare the method with "heavy duty" theorem proving in the spirit of [6]. However, some preliminary points for discussion have already emerged. Some of the shortcomings are reminiscent of those reported in [5]: the limitation to interger arithmetic, for example, means that the counter proofs presented by the system are more informative than the safety proofs holding over a limited range. This is, however, compensated in our approach by departing from fully formal proofs and combining with a simulation analysis when (local) reasoning over reals is crucial to the property in question.

Our model of the heat process intentionally made several simplifications to fit an early experimental set up [14]. The interested reader may for example refer to a more complex model of heat exchangers in [13] where some of our restrictions are relaxed. The purpose of that paper is the illustration of a rich simulation language and only the plant part of the heat exchanger is subjected to validation by simulation.

It is also interesting to note that the size of the real ventilation subsystem, compared to the demo system, in the same format as the one discussed in section 3 (NP-Tools circuit), is 700 input variables and 500 output variables. Despite the seemingly large state space, the size of the reachable states set – as far as required for the types of properties mentioned – is small enough for practical purposes, even in the real system [11].

Further work in the other parts of the project, specially extensions to the Lucifer prototype are very interesting for enhancing our verification methodology and incorporation of our methods in the system development process.

Acknowledgements

This work was supported by the Esprit LTR project SYRF, the Swedish board for technical research (TFR), and the Swedish board for technical development (NUTEK).

References

1. A. Poigné and M. Morley and O. Maffeïs and L. Holenderski. The Synchronous Approach to Designing Reactive Systems . *Formal Methods in System Design*, 12(2):163–187, March 1998.
2. G. Berry. The Foundations of Esterel. In *Proofs, Languages and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1998. To appear.
3. L. Besnard, P. Bournai, T. Gautier, N. Halbwachs, S. Nadjm-Tehrani, and A. Ressouche. Design of a Multi-formalism Application and Distribution in a Data-flow Context: An Example. In *Proceedings of the 12th international Symposium on Languages for Intentional programming, Athens, June 1999*. World Scientific.
4. B. Carlson, M. Carlsson, and G. Stålmarck. NP(FD): A Proof System for Finite Domain Formulas. Technical report, Logikkonsult NP AB, Sweden, April 1997. Available from `http://www-verimag.imag.fr//SYNCHRONE/SYRF/HTML97/a321.html`.
5. W. Chan, R.J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J.D. Reese. Model Checking Large Software Specifications. *IEEE Transactions on Software Engineering*, 24:498–519, July 1998.
6. B. Dutertre and V. Stavridou. Formal Requirements Analysis of an Avionics Control System. *IEEE Transactions on Software Engineering*, 25(5):267–278, May 1997.
7. N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.
8. D. Harel. STATECHARTS: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.
9. M. Heimdahl and N. Leveson. Completeness and Consistency in Heirarchical State-based Requirements. *IEEE transactions on Software Engineering*, 22(6):363–377, June 1996.
10. Integrated Systems Inc. *SystemBuild v 5.0 User's Guide*. Santa Clara, CA, USA, 1997.
11. O. Åkerlund. Application of Formal Methods for Analysis of the Demo System and parts of the Ventilation System of JAS 39 (in swedish). Technical report, Saab Aerospace AB, Linköping, Sweden, January 1997.
12. F. Maraninchi and Y. Rémond. Mode-automata: About modes and states for reactive systems. In *Programming Languages and Systems, Proceedings of the 7th European Symposium On Programming, Held as part of ETAPS'98, Lisbon, Portugal, LNCS 1381*. Springer verlag, March 1998.
13. S.E. Mattsson. On modelling of heat exchangers in modelica. In *Proc. 9th European Simulation Symposium*, Passau, Germany, October 1997. Currently available through `http://www.modelica.org/papers/papers.shtml`.

14. S. Nadjm-Tehrani. Integration of Analog and Discrete Synchronous Design. In *Hybrid Systems: Computation and Control, Proceedings of the second international workshop, March 1999, LNCS 1569*, pages 193–208. Springer Verlag, March 1999.
15. S. Nadjm-Tehrani. Time-Deterministic Hybrid Transition Systems. In *Hybrid Systems V, Proceedings of the fifth international workshop on hybrid systems, September 1997, LNCS 1567*, pages 238–250. Springer Verlag, 1999.
16. N. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *Proc. 11th International Conference on Automated Deduction, LNCS 607*. Springer Verlag, 1992.
17. The SYRF Project. Deliverables for Work packages 1 to 7. Available from `http://www-verimag.imag.fr//SYNCHRONE/SYRF/deliv1.html`, 1997-99.
18. Prover Technology. *NPTools v 2.3 User's Guide*. Stockholm, Sweden. Contact: `http://www.prover.com`.