SAFE CONTROLLERS DESIGN FOR INDUSTRIAL AUTOMATION SYSTEMS

José Machado*, Eurico Seabra*, José Campos***, Filomena Soares**, Celina Leão****

**Mechanical Engineering Department, **Electronics Engineering Department; University of Minho, School of Engineering; Campus of Azurém, 4800-058 Guimarães, PORTUGAL*

****Informatics Department, ****Industrial Engineering Department; University of Minho, School of Engineering; Campus of Gualtar, 4700-024 Braga, PORTUGAL*

Abstract: The design of safe industrial controllers is one of the most important domains related with Automation Systems research. For this purpose, there are used some important synthesis and analysis techniques. Among the analysis techniques two of the most important are Simulation and Formal Verification. In this paper these two techniques are used together on a complementary way. For the successful application of these mentioned techniques the plant modelling is crucial, so the understanding and modelling of the plant behaviour is essential for obtaining safe industrial systems controllers. The paper focuses on the plant modelling and its impact for Simulation and formal verification of real-time Industrial Controllers. A two step approach is used: first, the use of Simulation and, second, the use of Formal Verification of Industrial Systems Specifications. The specification and plant models used for each technique are described. Simulation and formal verification results are presented and discussed. The approach presented in the paper makes it possible to deal with real industrial systems, and obtain safe controllers for hybrid plants. This paper puts forward a systematized approach for safe controllers design, based on Simulation and Formal Verification techniques. Modelica modelling language and Dymola simulation environment is used for Simulation purposes and Timed Automata formalism and UPPAAL real-time model-checker are used for Formal Verification purposes.

Keywords: Industrial Systems Behaviour Modelling, Safe Controllers, Simulation, Formal Verification, Real-Time Systems

SAFE CONTROLLERS DESIGN FOR INDUSTRIAL AUTOMATION SYSTEMS

Abstract: The design of safe industrial controllers is one of the most important domains related with Automation Systems research. For this purpose, there are used some important synthesis and analysis techniques. Among the analysis techniques two of the most important are Simulation and Formal Verification. In this paper these two techniques are used together on a complementary way. For the successful application of these mentioned techniques the plant modelling is crucial, so the understanding and modelling of the plant behaviour is essential for obtaining safe industrial systems controllers. The paper focuses on the plant modelling and its impact for Simulation and formal verification of real-time Industrial Controllers. A two step approach is used: first, the use of Simulation and, second, the use of Formal Verification of Industrial Systems Specifications. The specification and plant models used for each technique are described. Simulation and formal verification results are presented and discussed. The approach presented in the paper makes it possible to deal with real industrial systems, and obtain safe controllers for hybrid plants. This paper puts forward a systematized approach for safe controllers design, based on Simulation and Formal Verification techniques. Modelica modelling language and Dymola simulation environment is used for Simulation purposes and Timed Automata formalism and UPPAAL real-time model-checker are used for Formal Verification purposes.

Keywords: Industrial Systems Behaviour Modelling, Safe Controllers, Simulation, Formal Verification, Real-Time Systems

1. Introduction

Modern engineered systems have reached a high degree of complexity that requires systematic design methodologies, and model-based approaches to ensure correct and competitive performance. Regarding the use of digital controllers, in particular, evidence shows that small errors in their design may lead to catastrophic failures.

Recent years have witnessed a significant growth of interest in the modelling, simulation and formal verification of physical systems. A key factor in this growth was the development of efficient equation-based simulation languages, and formalisms and tools able to deal with the combinatory explosion of discrete systems states.

Safety control has, as its main goal, the assurance of the reliability, availability, and maintainability requirements of automation systems. Because of its direct impact on people and goods safety, the reliability of critical systems (transports, space, nuclear, among others) has, since some time, mobilized the scientific community efforts. Assuring system safety, demands the use of a global approach, guaranteeing that weaknesses do not exist. This approach must take into account, first the set of engineering activities used during development, and later, after entering into operation, the set of activities of operational

exploitation and maintenance. Thus, there is a continuous effort, within the scientific community, to develop methods and tools enabling the anticipation of the possibility of malfunctioning, and to study how these different methods and tools might complement each other in the context of such a global approach.

Among the several techniques for industrial controllers' analysis, there are, distinguished by their utility, Simulation (Baresi et al. 2000) and Formal Verification (Moon, 1994). In the literature on industrial controller's analysis, these two techniques are rarely used simultaneously. If Simulation is faster to execute, it has the limitation of considering only a subset of all the system behaviour evolution scenarios. Using Formal Verification has the advantage of analyzing all the possible evolution scenarios but it presents some limitations on the dimension and complexity of the models that can be analyzed. Such limitations arise due to the time and computational resources that become necessary for the attainment of formal verification results for very large models. This paper shows, how it is possible, and desirable, to conciliate these two techniques in the analysis of industrial controllers. With the simultaneous use of these two techniques, the developed industrial controllers are more robust and not subject to errors. This paper is focused on the formal verification of real-time systems and also considers important aspects on the plant modelling tasks. The modelling of the plant is crucial for the development of safe controllers (Yalcin and Namballa, 2005).

There are several approaches to applying formal verification techniques on automation systems dependability: from formal verification by theorem proving (Roussel and Denis, 2002) to formal verification by model-checking (Rossi, 2004); and considering (Machado et al. 2006-a), or not (Rossi, 2004), a plant model. The above approaches, however, do not consider timed aspects. In the formal verification of timed systems, several approaches can be used to increase the quality of the verification results. A distinction can be made between the work by (Remelhe et al. 2004), where the translation of IEC 61131-3 Sequential Function Charts (SFC) into timed automata (Alur and Dill, 1990) is studied, and the work by (Gaid et al. 2005) where an approach for constructing the controller model is proposed. This latter work uses the initial approach proposed by (Mader and Wupper, 1999), and some work hypotheses related to the evolution of the controller model and time aspects, in order to increase the quality of the formal verification tasks' results.

In the present work the base system and the work hypothesis considered by (Gaid et al. 2005) for the controller behaviour are adopted, and we propose an approach to build timed plant models.

The paper is organized as follows. Section 1 presents the challenge being addressed in this work. Section 2 presents an overview of the most used analysis techniques for Industrial Controllers design, with special focus on Simulation and Formal Verification. Section 3 presents the impact of plant modelling on Simulation and Formal Verification analysis techniques (the most important analysis techniques for industrial controllers' design). Next, Section 4 is devoted to the general presentation of the case study, involving a system with two tanks, a heating device, level control sensors, and valves to control the liquids' flow. The specification for the controller, and the methodology used to deduce the controller program from the specification of the system's desired behaviour, is also presented in Section 4. Section 5 is

entirely devoted to plant modelling for simulation and formal verification purposes. Particular aspects that must be taken into account during the plant modelling tasks are highlighted. Next, in Section 6, a set of properties over system's behaviour is provided, as well as their formalization using Timed Computation Tree Logic (TCTL). Section 7 presents and discusses the results obtained from simulation and formal verification of the case study's specification, considering the models of the plant. Section 8 presents our systematized approach using both Simulation and Formal Verification. Additionally, it is shown how Plant modelling is essential in the adopted approach, and how to proceed in order to obtain safe controllers for automation systems. Finally, Section 9, presents conclusions and future work.

2. Analysis Techniques of Industrial Controllers Software

This section discusses the two main classes of approaches: Simulation and Formal verification, as applied to Industrial Controllers' Software.

2.1. Simulation

In the field of engineering, Simulation is used with different objectives and considering different software tools (Hlupic, 1999): from the simulation of mechanical systems behaviour, simulation of process behaviour (Huda and Chung, 2002), simulation of integrating manufacturing plants (Eben-Chaimea et al. 2004) to the simulation of more complex systems' behaviours, more precisely the Simulation of software for the control of automation systems. In this last case, several approaches are possible but the final goal is always to avoid major damages, and to be sure, before the realization of the controller, that the system will comply with expected behaviour.

Mathematical modelling and simulation are emerging as key technologies in engineering. Relevant computerized tools, suitable for integration with traditional design methods, are essential to meet future needs of efficient engineering. A number of approaches can be used to improve this technique, in order to obtain more accurate simulation results.

There is a large amount of simulation software, on the market, for automation systems design improvement. All languages and model representations are proprietary and developed for specific tools. There are general-purpose tools such as ACSL, SIMULINK and System Build. They are based on the same modelling methodology, input-output blocks, as used in the previous standardization effort, CSSL, from 1967 (Strauss, 1967). There are domain-oriented packages covering, for example, electronic programs (SPICE, Saber), multibody systems (ADAMS, DADS, SIMPACK), or chemical processes (ASPEN Plus, SpeedUp). With few exceptions, all such simulation packages are only strong in one domain, and are not capable of reasonably modelling components in other domains. This is a major disadvantage since technical systems are becoming more and more heterogeneous with components from many engineering domains.

Techniques for general-purpose physical modelling have been developed during the last decades, but did not receive much attention from the simulation market. Modern approaches build on non-causal modelling with true equations, and the use of object-oriented constructs to facilitate reuse of modelling knowledge. There are already several modelling languages with

such a support available from universities and small companies. Examples of such modelling languages include ASCEND (Piela et al. 1991), Modelica (Elmqvist et al. 1996), gPROMS (Barton and Pantelides, 1994), NMF (Sahlin et al. 1996), ObjectMath (Fritzson et al. 1995), Omola (Mattsson et al. 1993), SIDOPS+ (Breunese and Broenink, 1997), Smile (Kloas et al. 1995), U.L.M. (Jeandel et al. 1996) and VHDL-AMS (IEEE, 1997).

2.2. Formal Verification

As the complexity of the systems being built increases, so does the need for techniques and tools that allow analysis of their correctness and fit for purpose. In this context, it is commonly argued that formal mathematical notations should be used to support modelling and reasoning from the early stages of design and development (Jones, 1980). Using them, a model of the intended design can be developed and reasoned about.

Once a system model has been proposed, the question arises of determining its quality. Will the modelled system work as expected, and without incidents? Proving that the model is correct is not mathematically possible (Jones, 2003). This springs from the fact that, to prove it, some representation of the intended system against which to compare the model would be needed. But the model is *the* representation of the intended system.

Not being able to prove the correctness of the model is, nevertheless, different from not being able to reason about its quality. It is still possible to identify desirable properties of the system (measures of quality), and challenge the model with theorems expressing those properties. If the theorems can be proved, then it has been verified that the model exhibits those properties. If the modelling and proofs are done in some formal, mathematically based notation, then the verification is said to be formal.

This process of exploring the model with theorems representing properties to be verified is called formal specification verification (or validation). This is clearly different from formal program verification, the process of formally proving that a given system satisfies a specification, which was the traditional area of verification (Loeckx and Sieber, 1984) (Jones, 2003).

At this point it should also be clear that formal verification is different from simulation and testing. Formal verification establishes the validity of a property in a given specification in an absolute manner. Testing, in all but the simplest specifications, can only establish the validity of a property in a subset of the specification. While an appropriate choice of test cases can give a high degree of confidence, that confidence is never absolute.

The process of reasoning about formal models of systems has since long been an object of study (Jones, 2003). For a long time, however, formal proofs tended to stay in the area of envisaged benefits that were never actually completely fulfilled. It would be said that formal proofs could be done, but little was shown about how to actually prove interesting properties of a system. The fact is that formal proof tends to be a delicate, detailed, and time consuming process. As the complexity of models grows, tackling such proofs by hand becomes

increasingly harder. This has led to the study of mechanical reasoning techniques as a way to (at least partially) automate the analysis.

Two main categories of methods can be identified:

- deductive methods (i.e. theorem proving): these are semi-automated methods where a traditional mathematical proof is performed by a tool under user guidance – examples of theorem provers are PVS (Crow at al. 1995) and the Larch system (Guttag, 1993), but many others are currently available;

- algorithmic methods (i.e. model checking – see (Clarke et al. 1986)): these are fully automated methods and, given suitable system descriptions and properties, are capable of determining if a property is valid in a system, without human intervention.

Theorem provers take a deductive approach to verification. Proofs are performed in the traditional mathematical style, using some formal deductive system (a set of logical axioms, together with a set of deduction rules). Proofs progress by transforming (rewriting) a set of premises into a desired conclusion. The axioms and the deduction rules are the basis for the rewriting process.

Theorem provers provide a degree of automation. Nevertheless, it is still up to the verifier to decide the proof strategy but for the simplest proofs. Additionally, automated proofs need to be much more detailed than normal hand produced proofs, since no leaps in reasoning can be made. The prover must be ``convinced'' of the correctness of every step in the proof. So, automated theorem proving can be difficult to use. Learning curves, in particular, can be quite steep. Given these limitations, research in the area is continuing and alternative approaches have been sought.

Model checking was originally proposed as an alternative to the use of theorem provers in concurrent program verification in (Clarke et al. 1986). The basic premise was that a finite state machine specification of a system can be subject to exhaustive analysis of its entire state space to determine what properties hold of the system's behaviour. Typically the properties are expressed in some temporal logic that allows reasoning over the possible execution paths of the system. In this context, the possible execution paths are interpreted as alternative futures.

The kind of analysis that model checking is concerned with has basically to do with testing the *universality*, *inevitability* or *possibility* of given properties expressed in some temporal logic. For example, whether it can be guaranteed that the system will not be in a deadlock situation, or that users will have to save their work before quitting.

By using an algorithm to perform the state space analysis, the two main drawbacks of theorem provers were avoided:

- the analysis is fully automated (as opposed to theorem provers' high reliance on the skills of its users);

- the validity of a property is always decidable (as opposed to theorem provers' undecidability problems).

A main drawback of model checking has to do with the size of the finite state machine needed to specify a given system: useful specifications may generate state spaces so large that it becomes impractical to analyse the entire state space. Hence, theoretically decidable systems may become undecidable in practice. The use of Symbolic Model Checking (Burch, 1990) somewhat diminishes this problem. Avoiding the explicit representation of states, state spaces as big as 10^{20} states may be analysed.

As the maturity of the verification technology evolved, the feasibility of applying it in a more generalised manner ensued. As a result, in recent years several approaches to applying formal verification techniques on automation systems dependability have been proposed. These range from formal verification by theorem proving (Roussel and Denis, 2002) to formal verification by model checking (Rossi, 2004) (Gaid et al. 2005) (Machado, 2006).

At the same time, the success of the technique means that research has continued. Currently there are tools capable of dealing with richer models. For example, including issues of time, c.f. timed model checking, (Behrmann et al.2001), or probabilistic reasoning, c.f. stochastic model checking (Kwiatkowska et al. 2007). Timed model checking has been used, for example, by (Remelhe et al. 2004) and it used in the current paper.

3. Impact of Industrial Plants Modelling on Controllers' analysis techniques

This section presents the "state of the art" concerning different contexts of plant modelling, and its use in the context of industrial controllers' analysis techniques. The need to improve the safety of automation systems' behaviour is the major aspect that leads to the use of plant models.

3.1. Controllers analysis: Simulation

Simulation is an analysis technique that allows experimenting with a reduced and finite number of evolution scenarios of automation systems behaviour. While the results thus obtained are valuable for the tested scenarios only, it becomes possible to very quickly detect some errors in the specification of the controller. A number of simulation related works consider, in a more or less detailed manner, a plant model. See, for instance, the works of:

- (Baresi et al. 1998) (Baresi et al. 2000) where the plant is modelled by Simulink blocks,
- (Amerongen, 2003) where the plant is modelled by bond graphs,
- (Barton and Pantelides, 1994) where the plant is "divided" in small modules, with each module modelled by a finite state machine,
- (Mattsson et al. 1998) (Elmqvist et al. 1999) where the plant is modelled in the Modelica language,

- (Liu, 2004) where SPOA (System Performance-Oriented Automation), a plant modelling methodology, is proposed.

In all these works, the plant model is generally used for obtaining realistic evolution scenarios of the controller model execution. More important than the generation of numerous evolutions of the system (changing different logical inputs), it is preferable to obtain these evolutions from the evolution of a plant model. Following this methodology, the plant model has a direct influence (Gouyon, 2001) in the pertinence of the stimuli of the controller model and, clearly, in the pertinence of the results obtained with the simulation technique.

3.2. Controllers analysis: Formal Verification

The use of formal methods on industrial automation systems controllers verification may be classified on three levels, taking into account three different criteria (Frey and Litz 2000) :

- The used method: Model-checking (Lamperiere-Couffin et al. 1999), Theorem-proving (Roussel and Denis, 2002), Reachability analysis (Shanmugham and Roberts, 1995).
- The adopted formalism: Petri Nets (D'Souza and Khator, 1997), Net Condition/Event Systems (Rausch and Krogh, 1998), Finite state machines (Hassapis et al.1998).
- The use (or not) of a plant model:
    - Non model-based, without considering a plant model (Rossi, 2004);
    - Constrained-based, considering only some behaviour constraints (rudimentary model) (Canet et al. 2000);
    - Model-based, considering a real plant model, elaborated using a well defined formalism (Kowalewski and Preuβig, 1996). This model can be more or less refined depending on the behaviour properties that one intendss to prove.

Formal verification is an exhaustive technique. However, it is also a time consuming technique when compared, for instance, to simulation, and especially when a real case is being analysed. This is due to the complexity of the calculations performed. Also, it might happen that obtaining a solution is not feasible.

The utilization of a plant models, in formal verification tasks, will be studied in detail. An automation system is always composed by a controller coupled with a plant (Fig. 1). The controller outputs are the plant inputs, and the plant outputs are controller inputs.

Fig. 1. An automation system composed by a controller and a plant, connected.

Many works are focused on the formal verification of industrial controllers without considering the plant modelling. Among them, the most significant are (Bornot et al. 2000) (Lamperiere-Couffin et al. 1999) (Mertke and Frey 2001) (Rossi 2004). In these works plant models were not taken into account.

There are other works that, although not considering an explicit plant model, consider the introduction of some system behaviour constraints, and have thus improved considerably the obtained results (Canet et al. 2000).

On other works still, the plant model was considered in an explicit way. Among them, the most significant are (Rausch and Krogh, 1998) (Hassapis et al. 1998) (Kowalewski and Preuβig 1996) (D'Souza and Khator, 1997) where the plant is modelled with the utilization of the following formalisms: Petri Nets, Finite state machines and Net condition/event systems.

From a general point of view, plant modelling is done using a monolithic approach, and the plants that are modelled are small, when compared to the complexity of a standard industrial system. When the plant is modelled using a modular approach (Zaytoon and Carré-Ménétrier, 1999), the global model of the plant is typically obtained from the Cartesian product of the modules that compose it. This global model – even for a simple case study – becomes complex and with a higher dimension (number of states and transitions) leading to situations and states that do not have a physical signification.

In the works studied, when a plant model is used, a reduction of the reachable states of the controller model is performed. With the restriction imposed by the plant model some states of the obtained global model are not reached because the plant behaviour model imposes some restrictions of the controller model evolution. In this case, there are some behaviour properties that can be proved when the plant model is taken into account. Otherwise, it would be impossible to prove them.

Currently, plant models used in formal verification tasks are seen as a simple improvement in order to facilitate the performance of some formal verification techniques and tools like, for instance, some model-checkers.

Other important aspect that must be taken into account is the detail of the plant model considered. In fact, this factor directly affects the global model obtained for the automation system (a higher number of states has direct influence on the global computation time). The kind of properties that become possible to prove is directly related to the plant model's level of detail.

4. Case Study

In this work, a modified version of the benchmark example for an evaporator system, presented by (Kowalewski et al. 2001) and (Huuck et al. 2001), is used. The system (Fig. 2) consists of two tanks (tank1 is heated and mixed), a condenser, level sensors and on-off valves (Vi). In normal operation mode the system works as follows. Tank1 is filled with two solutions by opening valves V1 and V2. Then, the mixer starts working in order to promote the dilution. After two time units, the heated device is switched on for 20 time units to increase temperature solution. During this period part of the liquid is evaporated and cooled by the condenser. At that point the required liquid concentration has been reached and the heater is switched off. The remaining liquid is drained to tank2 by opening valve V3. The mixing device is switched off when tank1 is empty. The solution stays in tank2 for post-processing for 32 time units (to stay liquid), and then valve V4 is open to empty tank2.

Fig. 2. Evaporator system. Closed-loop system composed by controller and plant.

Throughout normal operation mode, the system may malfunction. During evaporation, the condenser may fail. In that case, the steam cannot be cooled and the pressure inside the condenser will rise. Therefore, the heater must be switched off to avoid the condenser explosion. By doing so, the temperature of tank1 will decreases and the solution may become solid, thus preventing drainage of tank2. Hence, valve V3 must be opened early enough, but only after opening valve V4, to prevent tank2 from overflowing.

In the case of a condenser malfunction, we also need to guarantee certain response times of the control program, taking into account the timing characteristics of the physical devices:

- whenever a condenser malfunction starts, the condenser can explode if steam is produced during 22 time units;
- if the heating device is switched off, the steam production stops after 12 time units;
- if no steam is produced in tank 1, the solution may solidify after 19 time units;
- emptying tank 2 takes between 0 and 26 time units;
- emptying tank 1 is very fast with respect to the other durations, so that it is considered as instantaneous;
- filling tank 1 takes at most 6 time units.

4.1. Controller specification

As we intended to conciliate the use of Simulation and Formal Verification tools, we needed to adopt a controller specification that could act as the basis of the controller program in the two analysis techniques. With that in mind, the controller specification was developed in IEC 60848 SFC because it can then be used as both the starting point for the development of the timed automata based Programmable Logic Controller program (PLC), to be verified with UPPAAL, and also the starting point for the development of the controller program to be used by the StateGraphs Modelica library (Otter et al. 2005).

The input and output variables of the controller model are summarized in Table 1; minimum and maximum level sensors and malfunction sensor are considered as inputs and on-off valves and Heater, Mixer and Alarm are controller program outputs.

Table 1.Input and output variables of the controller program.

Fig. 3. IEC 60848 SFC specification of the controller program.

4.2. Algebraic representation of the controller specification

As the basis for the controller modelling for simulation and formal verification purposes must be the same, the controller specification is translated to algebraic equations. These will be the basis for the controller models used on these two analysis techniques.

The desired behaviour for the described system was modelled using IEC 60848 SFC. The specification is presented in Fig. 3 . As IEC 60848 SFC is a specification language (and not a programming one), it becomes necessary to translate the SFC specification, first into a StateGraph program, and second into a program written in a PLC programming language (in this case the ladder language will be used). The translation from SFC to StateGraph is presented in (Seabra et al. 2007).

The goal of this work is to obtain a controller specification that, after being validated, will be implemented on a PLC (Programmable Logic Controller). So, the controller specification model takes into account the PLC behaviour (illustrated in Fig. 4).

Fig. 4. Cyclic scan monitor of the PLC

Let CC(q) (Clearing Condition) a Boolean variable associated to each transition of a SFC. A transition q (Fig. 5) can be cleared if it is enabled (all the steps that precede immediately this transition are active) and if its associated transition condition TC(q) is true. So, in a general case CC(q) can be formulated as follows:

$$CC(q) = (\prod_{j=1}^{m} Xj) \times TC(q) \quad (1)$$

with:

- Xj: step Boolean variable associated to step j,
- TC(q): Transition Condition associated to the transition q,
- m : number of steps that precede immediately step j.

Fig. 5. Transition condition after simultaneous sequences

According to the IEC 60848 evolution rules, the Boolean step variable Xi associated to each SFC step i can be computed in the following manner:

with:

- Xi(t): value of the step variable of step i for the tth scan cycle,
- Xi(t+1): value of the step variable of step i for the (t+1)th scan cycle,
- p: number of transitions that precede step i,
- n: number of transitions that follow step i,
- CC(pj): Clearing Condition of the transition (pj),
- CC(nk): Clearing Condition of the transition (nk).

The step activation/de-activation is done by:

$$Xi(t+1) = \sum_{j=1}^{p} CC(pj) + Xi(t) \times \prod_{k=1}^{n} \overline{CC(nk)} \quad (2)$$

In the case of step 2 of the above SFC, for instance, it comes then:

- $CC(9) = X11 \times T2E$ (3)
- $CC(10) = X12 \times T1E$ (4)
- $\mathrm{X12(t+1)} = CC(9) + X12(t) \times \overline{CC(10)}$ (5)

Computation of actions:

Each action is set when the logical OR of the step variables of the steps to which this action is associated is true. For instance:

- $V1(t) = X1(t)$ (6)
- $V1(t) = X11(t) + X12(t) + X13(t) + X21(t)$ (7)

With all the rules presented above, we construct the Ladder program based on the deduced equations. This is achieved by applying all the described steps to the controller specification presented in Fig. 3.

5. Plant Modelling

The interesting point about the plant modelling process was that it consisted of two steps: first, modelling the plant using the Dymola software and the Modelica programming language (Elmqvist and Mattson, 1997), and, second, using the Modelica models as the basis for the development of the UPPAAL models which are used on the formal verification tasks.

In the second step an "abstraction" is carried out from the hybrid models, obtained with Modelica, to the timed models that can be created with timed automata and verified with UPPAAL.

It would have been possible to directly translate the hybrid models, produced with Modelica, to hybrid models for Formal Verification purposes. However, the performance of the formal verification tasks, using hybrid systems or timed systems, is, of course, incomparable. The last one is faster and, using it, it is possible to treat standard industrial systems. Furthermore, most of the behaviour properties that typically we will intend to prove consist of properties that can be proved with timed Formal Verification. See (Campos et al. 2008) for a study of typical properties that can be found in the literature. There are other properties that we can only prove using hybrid formal verification. However, this aspect is not covered in this work.

5.1. Plant Modelling for Simulation purposes

5.1.1. Modelica Language

Modelica is a language for physical systems modelling that is being developed in the context of an international effort (Mattsson et al. 1997). It is suited for multi-domain modelling. For example, mechatronic models in robotics; automotive and aerospace applications involving mechanical, electrical, hydraulic and control subsystems; process oriented applications; and generation and distribution of electric power.

The design of Modelica builds on two relevant modern concepts in modelling and simulation. Namely, non–causal modelling and the use of object-oriented constructs (encapsulation, inheritance and hierarchy). Models in Modelica are mathematically described by differential, algebraic and discrete equations. The main objective is to make it easy to exchange models and model libraries.

Since Modelica accepts non-causal models, bond-graphs can be translated to Modelica code as sub-models (i.e. a-causally). Bond Graphs are a domain-independent graphical notion of physical systems modelling. During modelling, the edges in the graph denote the ideal exchange of energy between the sub-models (vertices). One can state that bond-graph modelling is in fact a form of object-oriented physical systems modelling.

In Modelica models and sub-models are declared as classes, with interfaces that are called connectors. A connector must contain all quantities needed to describe the interaction. Attributes can be used to specify how the connections are converted to computable code. Modification of a model definition is possible using the extended construct. This way, for refinement of a generic sub-model into a more specific one, only the 'new' specific parts need to be described. The common parts are inherited from the more generic sub-model, and need to be specified only once.

Modelica supports both high level modelling by composition and detailed library component modelling by equations. Models of standard components are typically available in models' libraries. Using a graphical model editor, a model can be defined by drawing a composition diagram (also called schematics), positioning icons that represent the models of the components, drawing connections and giving parameter values in dialogue boxes. Constructs for including graphical annotations in Modelica, make icons and composition diagrams portable between different tools.

5.1.2. Plant Model

All the system was modelled. Tank1 and tank2 models are presented on this sub-section. Considering the case of tank 1 we have, in the Modelica modelling language, the model presented in Figure 6.

Fig. 6. Modelica code for the tank1 model

The Modelica program code for modelling tank2, presented in Figure 7, is similar to the code obtained for the tank1 model. The main difference between these two codes is due to the tanks having different numbers of fill sources. Tank 1 has two fill sources, while tank 2 has sone.

Fig. 7. Modelica code for the tank2 model

5.2. Plant Modelling for Formal Verification Purposes

This sub-section starts with a description of the formalism adopted for plant modelling for formal verification purposes, and then introduces and discusses the plant model for the case study.

5.2.1. Modelling formalism

Timed automata were adopted as the modelling formalism for plant modelling due to two main reasons: first, the study of the proposed system needs to take time into account; and, second, it is the input formalism of the UPPAAL model-checker (Behrmann et al. 2004). Hence, is well adapted to the formal verification of timed systems.

Definition 1 (Timed Automaton (TA)). A timed automaton is a tuple (L, l0,C,A,E, I), where L is a set of locations, l0 $\in$ L is the initial location, C is the set of clocks, A is a set of actions, co-actions and the internal τ -action, E $\subseteq$ L × A × B(C) × 2^C × L is a set of edges between locations with an action, a guard and a set of clocks to be reset, and I : L $\rightarrow$ B(C) assigns invariants to locations. ¤

We now define the semantics of a timed automaton. A clock valuation is a function u : C $\rightarrow \Re_{\geq 0}$ from the set of clocks to the non-negative reals. Let $\Re^C$ be the set of all clock valuations. Let $u_0(x) = 0$ for all x $\in$ C. We will abuse the notation by considering guards and invariants as sets of clock valuations, writing u $\in$ I(l) to mean that u satisfies I(l).

Definition 2 (Semantics of TA). Let (L, l0,C,A,E, I) be a timed automaton. The semantics is defined as a labelled transition system $\langle S, s_o, \rightarrow \rangle$, where S $\subseteq$ L× $\Re^C$ is the set of states, $s_o = (l_o, u_o)$ is the initial state, and $\rightarrow \subseteq S \times \{\Re_{\geq 0} \cup A\} \times S$ is the transition relation such that:

– $(l,u) \xrightarrow{d} (l,u+d)$ *if* $\forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(l)$, *and*

– $(l,u) \xrightarrow{a} (l',u')$ *if there exists* $e = (l,a,g,r,l') \in E$ $s.t.$ $u \in g, u' = [r \rightarrow 0]u$, *and* $u' \in I(l)$,

Where, for d $\in \Re_{\geq 0}$, u + d maps each clock x in C to the value u(x) + d, and [r $\rightarrow$ 0]u denotes the clock valuation which maps each clock in r to 0 and agrees with u over C \ r. ¤

Timed automata are often composed into a network of timed automata over a common set of clocks and actions, consisting of n timed automata $A_i = (L_i, l_i^0, C, A, E_i, I_i)$, 1 ≤ i ≤ n. A location vector is a vector $\bar{l} = (l_1, \ldots, l_n)$.

We compose the invariant functions into a common function over location vectors $I(\bar{l}) = \Delta_i I_i(l_i)$. We write $\bar{l}\,[l_i'/l_i]$ to denote the vector where the ith element l_i of $\bar{l}$ is replaced by l_i'. In the following we define the semantics of a network of timed automata.

Definition 3 (Semantics of a network of Timed Automata). Let $A_i = (L_i, l_i^0, C, A, E_i, I_i)$ be a network of n timed automata. Let $\bar{l}_0 = (l_1^0, \ldots, l_n^0)$ be the initial location vector. The

semantics is defined as a transition system $\langle S, s_o, \rightarrow \rangle$, where $S = (L_1 \times \cdot \times L_n) \times \Re^C$ is the set of states, $s_0 = (\bar{l}_0, u_0)$ is the initial state, and $\rightarrow \subseteq S \times S$ is the transition relation defined by:

– $(\bar{l}, u) \rightarrow (\bar{l}, u + d)$ if $\forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(\bar{l})$

– $(\bar{l}, u) \rightarrow (\bar{l}\left[{}^{l_i'}/_{l_i}\right], u')$ if $there\ exists\ l_i \xrightarrow{\tau\ gr} (l_i',\ s.t.u \in g, u' = [r \rightarrow 0]u, and\ u' \in I(\bar{l}),$

– $(\bar{l}, u) \rightarrow \left(\bar{l}\left[{}^{l_j'}/_{l_j}, {}^{l_i'}/_{l_i}\right], u'\right)$ if $there\ exists\ l_i \xrightarrow{c?\ gr} l_i'\ and\ l_j \xrightarrow{c!\ g_j r_j} l_j'\ s.t.u \in (g_i \Delta g_j), u' = [r_i \cup r_j \rightarrow 0]u, and\ u' \in I(\bar{l}),$

5.2.2. Plant model

We consider the following eight modules for the present plant model: Tank1, Tank2, Heater, Mixer, Alarm, Steam, Condenser and Liquid.

It is worth noticing that there are two important aspects that we take into account:

- First, the set of simulation functioning delays are obtained by simulation. The obtained delays are, afterwards, used for the formal verification tasks, in order to define the time units used in the modules of the plant model; and
- Second, some behaviour properties that cannot be traduced for formal verification models (because they consider hybrid behaviour and we only consider formal verification of timed systems) can be simulated using Modelica and Dymola.

Model of tank1

The model of tank1 is, first, simulated with Dymola. The corresponding Modelica code is presented in Fig. 6.

The obtained delays on simulation were used on formal verification with UPPAAL. The corresponding model of the tank developed in UPPAAL for formal verification purposes, is presented in Fig. 8.

Fig. 8. UPPAAL model of tank1.

We consider four states:

- the empty state models tank1 being empty;
- the filling state models liquid entering in tank1;
- the full state models tank1 being full;
- state overflow is also considered; this is a possible state for the tank, but describes an undesired behaviour.

In this model, it is also considered that tank1 is emptied in a very short time, when compared with the filling time. Hence, we have considered this time to be null. It is for this reason that the model goes from the full state directly to the empty state, without an intermediate state. The Boolean variables T1E and T1F are associated with tank1.empty and tank1.full, respectively. These variables represent the level sensors' signals sent by the sensors from the plant to the controller. The maximum time for filling tank1 is six time units.

Model of tank2

The model of tank2 is similar to that of tank1, and the reasoning followed to obtain this model was the same as presented before for obtaining tank1's model. As empting tank1 is considered to take a short (null) time, the filling of the tank2 is done in the same conditions, since the liquid is transferred from tank1 to tank2. Four states are considered: empty, full, emptying and overflow, which (again) is a possible state for the tank, but describes an undesired behaviour. The variables T2E and T2F have the same behaviour on the model for tank2 as T1E and T1F described above for tank1. Emptying tank2 takes, at most, twenty-six time units (Fig. 6).

Fig. 9. UPPAAL model of tank2.

Models of the Heater, Mixer and Alarm

The reasoning adopted when building these three models was the same: for all of them it consisted in considering two states for each model:

- state off – the initial state for all of them, and;
- state on – indicating that the orders sent from the controller to the plant are active.

Fig. 10, Fig. 11 and Fig. 12 present the models for the Heater, Mixer and Alarm, respectively.

Fig. 10. Model of the Heater

Fig. 11. Model of the Mixer

Fig. 12. Model of the Alarm

Model of the condenser

The model of the condenser, presented in Fig. 13, is composed of five states:

- state good models the good functioning of the condenser;
- state malfunction_heater models that the condenser is malfunctioning and the heater is in its on state;
- state malfunction_not_heater models that the condenser is malfunctioning but the heater has been switched off;
- state before_explosion models the behaviour where it is not possible to avoid the condenser explosion; and

- state explosion models the behaviour of the condenser explosion.

Fig. 13. Model of the condenser

In the behaviour described in the case study, it is stated that the condenser may explode if it starts malfunctioning, and steam exists for the duration of twenty-two time units after that. In the model, if the system remains in the malfunction_heater state (the heater is in the on state) during ten time units, then the condenser will inevitably explode because we will have steam for more twelve time units (in state before_explosion). The malfunction behaviour happens in a random way from the state good of the condenser model: we have added to the condenser model a Boolean variable Malf that indicates that behaviour.

Model of the steam

One of the hardest tasks on plant modelling is to model plant products (as the case of steam) because these models are specific for each plant. In this case, and taking into account the above described behaviour for the system, the model of steam is extremely useful to allow us to prove some system behaviour properties. We thus consider three states for the model:

- state off models that steam does not exist;
- state on_heat_on models that steam exists and the heater is also in state on (the model of the steam is dependent of the model of the heater), and
- state on_heat_off models that steam exists but the heater is in the off state.

After the heater switches off, steam lasts for twelve time units. We have introduced the Boolean variable steam_var that indicates when steam exists or not.

Fig. 14. Model of the steam

Model of the liquid

As the steam model presented before, the model of the liquid (Fig. 15) is also a specific model related with a plant product.

Fig. 15. Model of the liquid

This model has been created from the view point of the presence of liquid in tank1, regarding, also, the possible liquid solidification.

From the point of view of the presence of liquid in tank1 (see Fig. 15) we have four possibilities:

- there is no liquid in tank1, modelled by state no_liquid_1;
- there is liquid in tank1 and the liquid is heating, modelled by state liquid_heat_on;
- there is liquid in tank1 and the liquid is not heating, modelled by state liquid_heat_off; and

- there is liquid in tank1 that changed to the solid state after thirty-one time units, modelled by state solid (these thirty-one time units mean that the liquid solidifies nineteen time units after the absence of steam; after the heater is switched off, the steam is present for more twelve time units).

We point out that, for all the models, we have adapted the simulation conditions to obtain the same delays proposed on (Gaid et al. 2005), in order to have a basis to compare the results of the formal verification.

6. System Behaviour Properties and Properties Formalization

Having a formal model of the controller program enables us to check if it exhibits the desirable properties regarding safe and correct operation. Before we can carry out the verification step in a model-checking tool, however, we need to:

- Decide what are the relevant properties the model should exhibit;
- Encode the properties in a logic suitable for automatic verification (in this case, model checking).

Regarding the first step above, in general we will be interested in guaranteeing that undesirable states of the process are not possible/reachable (safety), and that desirable states can be reached (liveness). What these desirable/undesirable states are is domain dependent. The exact nature of the process and of the controller greatly influences the choice of relevant properties.

In the current case, we have chosen seven properties that, besides being considered relevant in the context of the case study, are also examples of the type of results that can be achieved through verification. The properties are:

P1: The tank will always be full when the heater is working – violation of this condition can lead to tank 1 being damaged due to overheating.

P2: During the evaporation step, steam will leave tank 1 into the condenser only when the heater is on and the valves V1, V2 and V3 are closed.

P3: The input and output valves of a tank will never be simultaneously open – violation of this condition would lead to uncontrolled liquid flow.

P4: The condenser will never explode.

P5: The solution will never solidify in tank 1.

P6: Tank1 will never overflow.

P7: Tank2 will never overflow.

Once we have decided on the properties we want to check, we must choose an appropriate logic.

Fig. 16. A State machine and its Computation tree. Adapted from (Clarke et al. 1986).

It is important to choose a formal logic adapted to the verification technology being used. UPPAAL uses a simplified version of TCTL (Alur et al. 1993), a timed version of CTL (Computational Tree Logic). CTL (Clarke et al. 1986) is a propositional, branching-time temporal logic that enables expressing queries over the possible behaviours of a model. In CTL the behaviour of a system is seen as a tree representing behaviour alternatives (Fig. 16). CTL formulae consist of path formulae and state formulae. State formulae are evaluated in individual states. Path formulae quantify over paths in the behaviour of the model, and over the states in those paths.

In what follows it is enough to know that, in the UPPAAL version of the logic, A is the universal quantifier on paths (*for any path*…), and [ ] the universal quantifier over states in a path (*for any state*…). Hence, if p is a state formula, then the combination A[ ] p means *for all states in the future p holds*. State formulae are expressed in propositional logic with the usual operators.

Bearing in mind the models described in the previous section and the temporal logical operators just described, the properties identified above are formalized as follows:

P1: A[ ] (H imply T1F) – that is, it is always the case (A[]) that if the heater is on (H) then the tank will be full (T1F).

P2: A[ ] (H imply (not(V1) and not(V2) and not(V3))) – that is, it is always the case (A[]) that if the heater is on (H) then valves V1 to V3 will be closed (not(V1) and not(V2) and not(V3)).

P3: A[ ] not (((V1 or V2 or V4) and V3)) – that is, it is always the case (A[]) that valve V3 will not be open at the same time as either valves V1, V2 or V4.

P4: A[ ] not condenser_explosion – that is, it is always the case that the condenser_explosion condition will not hold.

P5: A[ ] not liquid_solid – that is, it is always the case that the liquid_solid condition will not hold.

P6: A[ ] not tank1_overflow – that is, it is always the case that the tank1_overflow condition will not hold.

P7: A[ ] not tank2_overflow – that is, it is always the case that the tank1.overflow condition will not hold.

Once the properties have been formally expressed, verification can be carried out. That step is addressed in section 7.2.

7. Obtained results

After the modelling of the studied system, the results of Simulation and Formal Verification are presented.

7.1. Simulation Results

In order to perform the simulation, it is necessary to define a number of parameters: start and stop time of the simulation, the interval output length or number of output intervals, and the integration algorithm. In the present work, in all simulations performed, the Dass algorithm (Basu et al. 2006) with 500 output intervals was used.

In order to study the system behaviour different values for physical variables of the plant were used. Table 2 shows the variables considered in the simulation of the system behaviour.

Table 2.Variables of the plant.

The first two simulation performed were devoted to verify if the SFC of the controller system (Fig. 3), modeled in the Modelica language with the StateGraph library for hierarchical state machines, correctly simulated the evaporator system, both in normal and malfunction operation. The values for the plant variables considered in these simulations were Q1=1, Q2=0.5, G1=G2=1, Ht1=Ht2=1, A1=0.2 and A2=0.05.

Fig. 17 shows results of the simulation without the occurrence of the condenser malfunction during the production cycle. This corresponds to the normal operation for the tanks' level and for the controller outputs. It can be also observed, in the same figure, that the system is properly simulated by the developed program, since during the time specified by the SFC the tanks remain filled and empty, as well as, the switch logical state of the controller outputs.

On the other hand, Fig. 18 shows results of the simulation with the occurrence of the condenser malfunction during the production cycle, which corresponds to the malfunction operation. The malfunction occurred in a random way 15s after the start of the plant functioning. Analyzing Figure 12 it can be concluded that the proposed program properly simulates malfunction operation. Because it can be verified, taking into account figure 12, that at the malfunction occurrence (time 15s) the solution present in the tank1 is immediately drained for the tank2 and later emptied. In the same way, analyzing Figure 12, it can be verified that at time 15s the switch off the mixer and the heater, and the alarm switch on, occur simultaneously. This matches to the SFC specification of the controller.

Fig. 17. Level tanks in function of time in normal operation of the evaporator system.

Fig. 18. Switch state of the mixer, heater and alarm in normal operation of the evaporator system.

Fig. 19. Level tanks in function of time with occurrence of condenser malfunction (time = 15s).

Fig. 20. Switch state of the mixer, heater and alarm with occurrence of condenser malfunction (time =15s).

It is also necessary, in addition to the verification that the modelling of the system conforms to the SFC of the system controller, to guarantee that in the case of condenser malfunction solidification or explosion of the solution in the tanks does not occur. Thus it is necessary to take into account the timing characteristics of the physical devices.

For example, in the simulations presented in Fig. 19 and Fig. 20 (which considered the following values for the plant variables: Q1=1, Q2=0.5, G1=G2=1, Ht1=Ht2=1, A1=0.2 and A2=0.05), the times obtained for filling and emptying tank1 were, respectively, 0.6533s (limit 6s) and 2,1255s (limit 19s), and for the tank 2, respectively, 2,2655s (similar to the time of empty tank1) and 8,4361s (limit 26s). This simulation allowed showing that, with these plant variables' values, the system does not have serious functioning anomalies that can put in risk humans lives and material goods.

In order to obtain the relation between the plant variables and the time of the critical operations, some simulations were performed using several values of plant variables. Figure 10 and 11 show results of these simulations, respectively related to the empty tanks time (equal for the tank1 and tank2) and fill tank1 time, which correspond at the times of the critical operations of the evaporator system.

Fig. 21. Empty tank time in function of plant variables.

Fig. 22. Fill tank1 time in function of plant variables.

7.2. Formal Verification Results

The strategy adopted for formal verification considers that the program is written in Ladder language (according to section 2) and the following work hypotheses, as presented on (Gaid et al. 2005):

- the PLC executes the control program with a three phases cyclic behaviour;
- IEC 61131-3 TON timer blocks are considered;
- the duration of PLC cycle is considered to be between ε1 and ε2 and we have adopted ε1 equal to 0 and ε2 equal to 1;
- the unit of time used for the plant is irrelevant;
- discovering the maximal value of ε2 for the properties to hold was not our objective.
- The controller and the plant models, coupled as a closed-loop system, compose the global model verified.

All the properties were verified in less than 2 seconds each, using UPPAAL version 4.0.3 and an Intel Core Duo, 1,87GHz, machine with 4GB of RAM. The modular approach to building the plant model is well suited to facilitate the tasks of writing the properties, as is the case of properties P4, P5, P6 and P7. These properties consider undesirable states of the plant modules' models and we verify if these undesirable states are reached or not. For instance, if the state explosion of the condenser is reached that means a damage and an undesirable

behaviour. This is expressed by the property P4: A[ ] not *condenser.explosion* where it is expressed that the explosion state of the condenser is never reached.

8. Proposed approach for safe controllers design

In order to obtain safe controllers we present a systematized approach, using the analysis techniques Simulation and Formal Verification, together, where Plant modelling plays a crucial role. The proposed approach has two main ideas: first the use of the analysis techniques (Simulation and Formal Verification) on a complementary way and, second, the use of well defined and formalized plant models, for the Simulation and Formal Verification tasks.

The approach is divided in two steps that when performed in sequence, are efficient in obtaining safe controllers. The two steps are described below.

First Simulation is used, and then Formal Verification is used. Fig. 23 presents the first step with the use of Simulation, and Fig. 24 presented how the Formal Verification is used after Simulation.

Fig. 23. Step 1 – Using Simulation with programming language Modelica and Dymola software.

Fig. 24. Step 2 – Using Formal Verification with timed automata and UPPAAL.

It is on the second step, and in the respective Plant modelling approach, that this paper is focused: obtaining the Timed Plant Model.

After these two steps we can be sure that our specification is safe and as we use an automatic translation of the specification to the program – based on the algebraic translation of the IEC 60848 SFC (Machado et. al 2006-b) - we can guarantee that the program is safe too.

Anyway, the methodology presented in this paper, for modelling the plant, is the same if it is intended to verify the specification or the program.

9. Conclusions

The development of timed Plant models (for Formal verification purposes) as an abstraction of hybrid models (used in Simulation) is very interesting from the point of view of the performance that can be obtained in the Formal Verification tasks. Nowadays, formal verification of timed systems allows us to deal with high dimension cases, enabling us to study complex real world cases.

Another important aspect of the work is that, with the presented approach to building plant models (using first Simulation and then Formal Verification), we can address two important points:

- Firstly, by using Simulation, we can avoid a number of program errors in a reduced time interval – this would not happen if these errors were detected only through the use of Formal Verification techniques.

- Secondly, our modular approach provides us with an easier way to build plant models which are well suited to facilitate the task of writing the properties formulae. In fact, the consideration of undesirable states in the created plant models simplifies the task of writing properties.

The use of the Modelica programming language, to obtain these modular plant models, is useful to define the delays in which a property can, or cannot, be proved and also the delays to elaborate the considered plant models.

simulation techniques allow us to test different delays of the plant functioning, and to see if a property, considering different delays, is always true; or if different delays imply that a property that is true, for a delay, will become false for another. This study is very important from the point of view of system safety and is part of the on-going work performed on this subject.

10. Acknowledgements

This research was carried out in the context of the SCAPS Project supported by FCT, the Portuguese Foundation for Science and Technology, and FEDER, the European regional development fund, under contract POCI/EME/61425/2004 that deals with safety control of automated production systems.

References

Alur R., C. Courcoubetis and D. L. Dill (1993). Model-Checking in Dense Real-Time, Information and Computation, vol. 104, n_ 1, pp. 2-34.

Alur R. and D. L. Dill (1990). Automata for modeling real-time systems. Proc. of 17th Int. Coll. Automata, Languages, and Programming (ICALP'90), England.

Amerongen J. V. (2003). Mechatronic Design. Mechatronics Vol. 13 pp. 1045–1066.

Baresi L., S. Carmeli, A. Monti, M. Pezzè (1998). PLC Programming Languages: A formal Approach. Automation 98, ANIPLA

Baresi L., M. Mauri, A. Monti and M. Pezzè (2000). PLCTOOLS: Design, Formal Validation, and Code Generation for Programmable Controllers. Simulation software.

Barton, P. and C. Pantelides (1994). Modeling of combined discrete/continuous processes. AIChE J., Vol.40, pp. 966–979.

Basu S., R. Pollack, M. Roy (2006). Algorithms in Real Algebraic Geometry - Algorithms and Computation in Mathematics. Springer Editions, vol. 10, 2nd edition.

Behrmann G., K. G. Larsen, O. Moller, A. David, P. Pettersson, W. Yi (2001). Uppaal - present and future," in Proc. of IEEE Conference on Decision and Control, pp. 2881-2886.

Behrmann G., A. David, K. G. Larsen (2004). A Tutorial on UPPAAL. In Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04). LNCS 3185

Bornot S., R. Huuck, B. Lukoschus (2000). Verification of Sequential Function Charts Using SMV, At the PDPTA 2000 special session on Formal Validation, Las Vegas, Nevada, USA, June 26-29.

Breunese, A. P. and J. F. Broenink (1997). Modeling mechatronic systems using the SIDOPS+ language. In Proceedings of ICBGM'97, 3rd International Conference on Bond Graph Modeling and Simulation, Simulation Series, Vol.29, No.1, pp. 301–306. The Society for Computer Simulation International.

Burch J., E. Clarke, K. Mcmillan, D. Dill, and L. Hwang (1990). Symbolic model checking: 10^20 states and beyond. In Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science. Washington, D.C.: IEEE Computer Society Press, pp. 1-33.

Campos, J. C., J. Machado, and E. Seabra (2008). Property Patterns for the Formal Verication of Automated Production Systems. Proceedings of IFAC'2008 – 17th IFAC World Congress, July 6-11th, Seoul, South Korea.

Canet G., S. Couffin, J.-J. Lesage, A. Petit, P. Schnoebelen (2000). Towards automatic verification of PLC programs written in Instruction List, Proceedings of the IEEE, SMC.

Clarke E. M., E.A. Emerson, A.P. Sistla (1986). Automatic verification of finite state concurrent systems using temporal logic specifications. In ACM Transactions on Programming Languages and Systems, Vol. 8 (2), pp. 244 - 263.

Crow J., S. Owre, J. Rushby, N. Shankar, and M. Srivas (1995). A Tutorial Introduction to PVS. In Workshop on Industrial-Strength Formal Specification Techniques.

David A., G. Behrmann, K. G. Larsen and W. Yi (2003). A Tool Architecture for the Next Generation of UPPAAL, Technical Report n. 2003-011, Uppsala University, February. 20 pages.

Eben-Chaimea M., N. Pliskina and D. Sosna (2004). An integrated architecture for simulation. Computers & Industrial Engineering, Volume 46, Issue 1, pp. 159-170.

Elmqvist E. and S. Mattson (1997). An Introduction to the Physical Modelling Language Modelica. Proceedings of the 9th European Simulation Symposium, ESS'97, Oct 19-23, Passau, Germany.

Elmqvist H., S. E. Mattsson, M. Otter, (2001): Object-Oriented and Hybrid Modeling in Modelica. Journal European Journal of Automated Systems, Vol. 35, No 1.

Frey G. and L. Litz (2000). Formal methods in PLC programming. Proceedings of the IEEE Conference on Systems, Man and Cybernetics, SMC 2000, Nashville, October 8-11.

Fritzson P., L. Viklund, D. Fritzson and J. Herber (1995). High-level mathematical modeling and programming. IEEE Software, Vol.12, No.3.

Gaid M., B. Bérard and O. Smet (2005). Verification of an evaporator system with UPPAAL. European Journal of Automated Systems, vol. 39, nº9, pp. 1079-1098.

Gouyon D. (2001). Application de techniques de sysnthèse de la commande en ingéniérie d'automatisation. MSc Thesis; University Henri Poincaré, Nancy I.

Guttag J. V. and J. J. Horning (1993). Larch: Languages and Tools for Formal Specification. New York, NY, USA: Springer-Verlag New York, Inc.

Hassapis G., I. Kotini, and Z. Doulgeri (1998). Validation of a control system software specified in SFC notation by using hybrid automata, INCOM'98, Vol.II, pp. 65-70.

Helmqvist, , H., F. Cellier, and M. Otter (1994). Object oriented modeling of power-electronic circuits using Dymola. In Proceedings of the First Joint Conference of International Simulation Societies (CISS). The Society for Computer Simulation, ETH, Zurich, Switzerland.

Hlupic V. (1999). Simulation Software: Users' Requirements. Computers & Industrial Engineering Vol. 37, Issues 1-2, pp. 185-188.

Huda A. and C. Chung (2002). Simulation modeling and analysis issues for high-speed combined continuous and discrete food industry manufacturing processes. Computers & Industrial Engineering, Vol. 43, Issue 3, pp. 473–483.

Huuck R., B. Lukoschus and Y. Lakhnech (2001). Verifying Untimed and Timed Aspects of the Experimental Batch Plant. European Journal of Control, vol. 7, n_ 4, p. 400-415.

IEEE (1997). Standard VHDL Analog and Mixed-Signal Extensions. Technical Report IEEE 1076.1. IEEE.

Jeandel, A., F. Boudaud, P. Ravier and A. Buhsing (1996). U.L.M: Un Langage de Modélisation, a modelling language. In Proceedings of the CESA'96 IMACS Multiconference. IMACS, Lille, France.

Jones, C. B. (1980). Software Development: a Rigorous Approach. Prentice Hall PTR

Jones, C.B. (2003) The early search for tractable ways of reasoning about programs," Annals of the History of Computing, IEEE , vol.25, no.2, pp. 26-49, April-June. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1203057&isnumber=27086

D'Souza K., and S. Khator (1997). A control model for detecting deadlocks in an automated machining cell. Computers & Industrial Engineering, Vol. 32, No. 2, pp. 455-465.

Kloas, M., V. Friesen and M. Simons (1995). Smile -A simulation environment for energy systems." In Sydow, Ed., Proceedings of the 5th International IMACS-Symposium on Systems Analysis and Simulation (SAS'95), Vol.18–19 of Systems Analysis Modelling Simulation, pp. 503–506. Gordon and Breach Publishers.

Kowalewski S., J. Preußig (1996). Verification of sequential controllers with timing functions for chemical processes, 13th IFAC World Congress, San Francisco, USA, 30 June - 5 July, invited session on Advances in Discrete Event Control.

Kowalewski S., O. Stursberg and N. Bauer (2001). An Experimental Batch Plant as a Test Case for the Verication of Hybrid Systems. European Journal of Control. vol. 7, n_4, pp. 400-415.

Kwiatkowska M., G. Norman and D. Parker (2007). Controller Dependability Analysis By Probabilistic Model Checking. Control Engineering Practice, 15(11), pp. 1427-1434.

Lampérière-Couffin S., O. Rossi, J.-M. Roussel, J.-J. Lesage (1999). Formal validation of PLC programs: A survey, Proceedings of the ECC'99, Karlsruhe Germany, August 1999, cdrom paper N° 741.

Liu Y. (2004). Modélisation et simulation pour l'automatisation orientée performances de systèmes de production. Post-Doc final report for the University Henri Poincaré, Nancy 1, session at 6 january 2004, pp. 11-27.

Loeckx J., K. Sieber. (1984). The Foundations of Program Verification. Wiley, Chichester, England.

Machado J. (2006). Influence de la prise en compte d'un modèle de processus en vérification formelle des Systèmes à Evénements Discrets. PhD Thesis, École Normale Supérieure de Cachan, France.

Machado J., B. Denis, J.-J. Lesage (2006-a). A generic approach to build plant models for DES verification purposes. Proc. of Wodes'2006 – 8th Workshop on Discrete Event Sistems. July, 10-12, Ann Arbor, Michigan, USA.

Machado, J., B. Denis, J-J. Lesage, J-M. Faure, J. C. L. Silva (2006-b). Logic Controllers dependability verification using a Plant Model. DESDes IFAC Conference, Poland, 18-20th September

Mader A. and H. Wupper (1999). Timed Automaton Models for Simple Programmable Logic Controllers. Proc. Of the 11th Euromicro Conference on Real-Time Systems (ECRTS'99), York, UK.

Mattsson, S. E., M. Andersson and K. J. Åström (1993). Object-oriented modelling and simulation. In Linkens, Ed., CAD for Control Systems, chapter 2, pp. 31–69. Marcel Dekker Inc, New York.

Mattsson, S.E., H.E. Elmqvist and J.F.Broenink (1997). Modelica™ - An international effort to design the next generation modelling language, Journal A 38, No.3, pp.16-19.

Mattsson S. E., H. Helmqvist, M. Otter (1998). Physical System Modelling with Modelica. Control Engineering Practice Vol. 6, pp. 501-510.

Mertke T., G. Frey (2001). Formal Verification of PLC-programs generated from signal interpreted Petri Nets, Proceedings of the SMC 2001, Tucson (AZ), pp. 2700-2705.

Moon I. (1994). Modeling programmable logic controllers for logic verification. IEEE Control Systems, vol. 14, nº2, pp. 53-59.

Otter M., K. Årzén and I. Dressler (2005). StateGraph - A Modelica Library for Hierarchical State Machines. In: Modelica 2005 Proceedings.

Piela P., T. Epperly, K. Westerberg and A. Westerberg (1991). ASCEND: An object-oriented computer environment for modelling and analysis: the modelling language. Computers and Chemical Engineering, Vol.15, No.1, pp. 53–72.

Rausch M., B. H. Krogh (1998). Formal Verification of PLC Programs, American Control Conference, Philadelphia, PA, USA.

Remelhe M., S. Lohmann, O. Stursberg and S. Engell (2004). Algorithmic Verification of logic Controllers given as Sequential Function Charts. Proceedings of the IEEE International Symposium on Computer Aided Control Systems Design, Taipei, Taiwan.

Roussel J-M. and B. Denis. (2002). Safety properties verification of ladder diagram programs. European Journal of Automated Systems, Vol. 36, pp. 905-917.

Rossi O. (2004). Validation formelle de programmes ladder pour automates programmables industriels. PhD thesis, June, France.

Sahlin, P., A. Bring and E.F.Sowell (1996). The Neutral Model Format for building simulation, Version 3.02. Technical Report. Department of Building Sciences. The Royal Institute of Technology, Stockholm, Sweden.

Seabra E., J. Machado, F. Soares, C. Leão, J. Silva (2007). Simulation and formal verification of real-time systems: A case study. In: ICINCO'2007 proceedings, 9-12th may, Angers, France.

Shanmugham S. and C. Roberts (1995). A Verification Methodology for Real-time Supervisory Control Specification. Computers & Industrial Engineering, Vol. 29, No. 1-4, pp. 705-709.

Strauss J. C. (1967). The SCi continuous system simulation language (CSSL). Simulation, Vol. 9, pp. 281–303.

Yalcin A., R. Namballa (2005). An object-oriented simulation framework for real-time control of automated flexible manufacturing systems. Computers & Industrial Engineering, Vol. 48, Issue 1, pp. 111-127

Zaytoon J., V. Carré-Ménétrier (1999). Grafcet et graphe d'états : comportement, raffinement, vérification et validation, APII – Journal Européen des Systèmes Automatisés, Vol. 7, pp. 751-782.

Table1

Click here to download high resolution image

Input	Output
start – process start level1 – % fill tank1 level2 – % fill tank2 malf – condenser malfunction	V1 – open valve1 V2 – open valve2 V3 – open valve3 V4 – open valve4 H – switch Heater on MR – switch Mixer on Alarm – start alarm

Table2

Plant	Variable
source1, source 2	Q1, Q2 - flow rate [m^3/s]
tank1, tank2	G1, G2 – ground area [m^2] Ht1, Ht2 –height [m] A1, A2 – drain hole area [m^2]

Figure1
Click here to download high resolution image

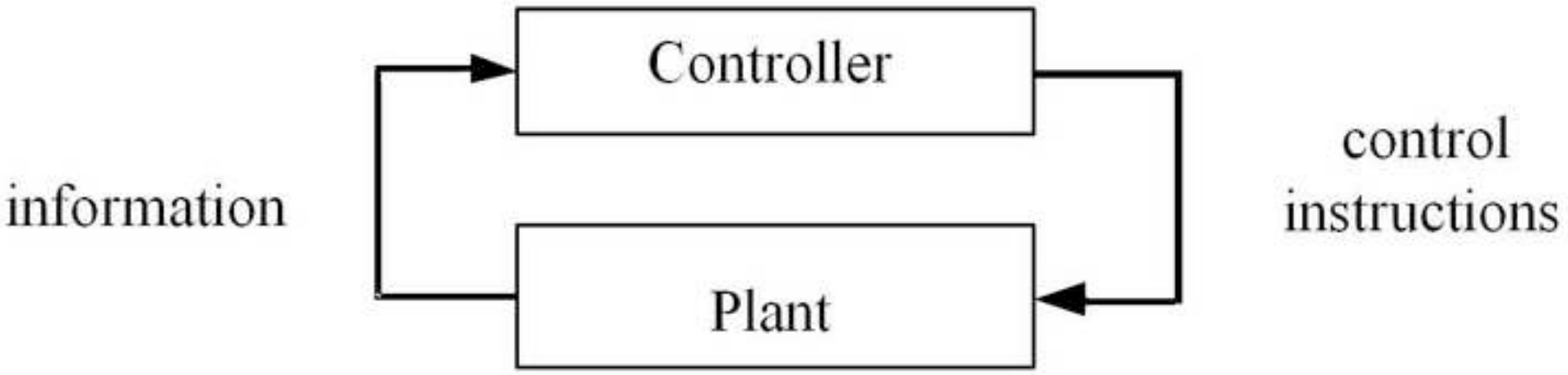

Figure2

Click here to download high resolution image

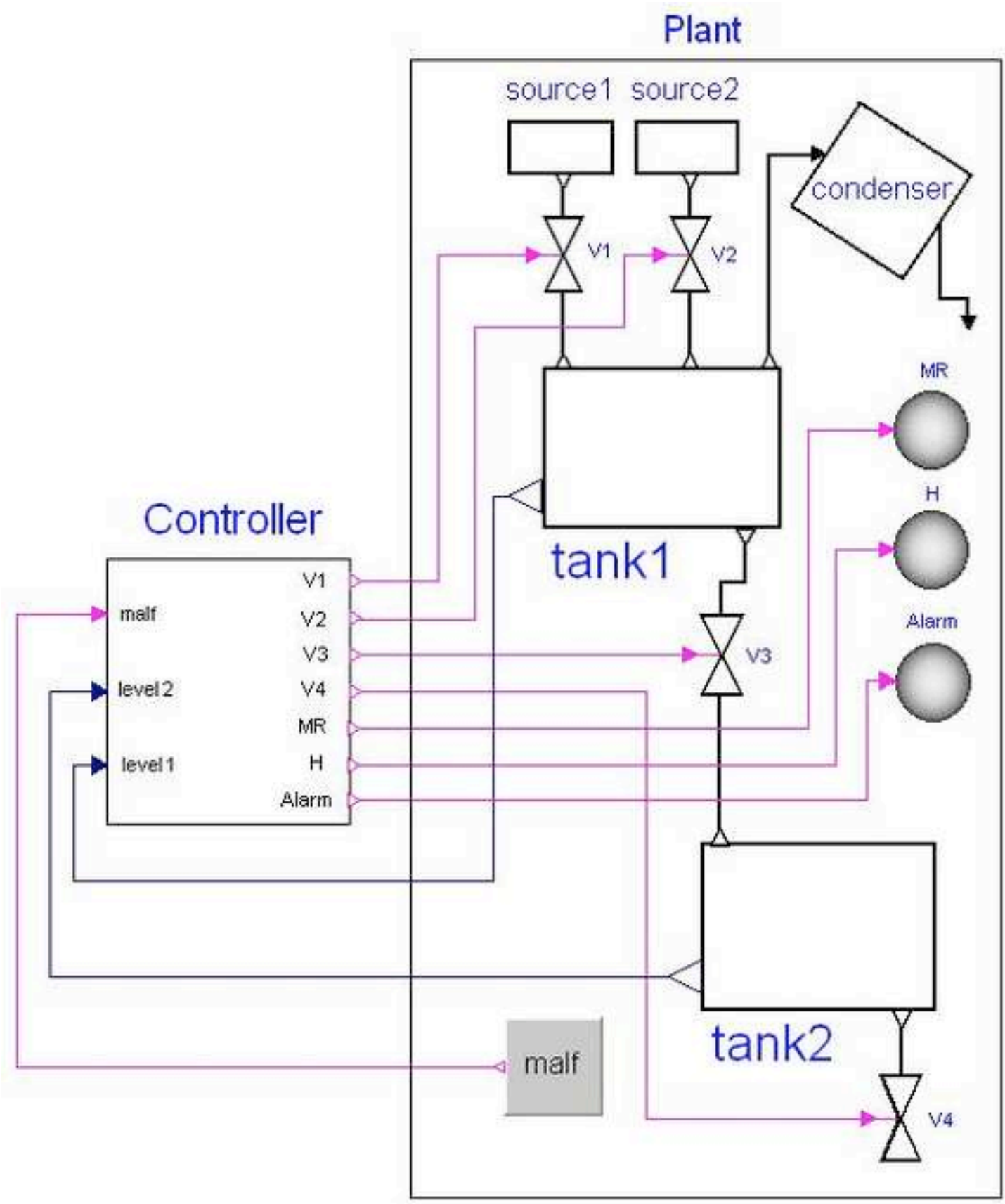

Figure3

Click here to download high resolution image

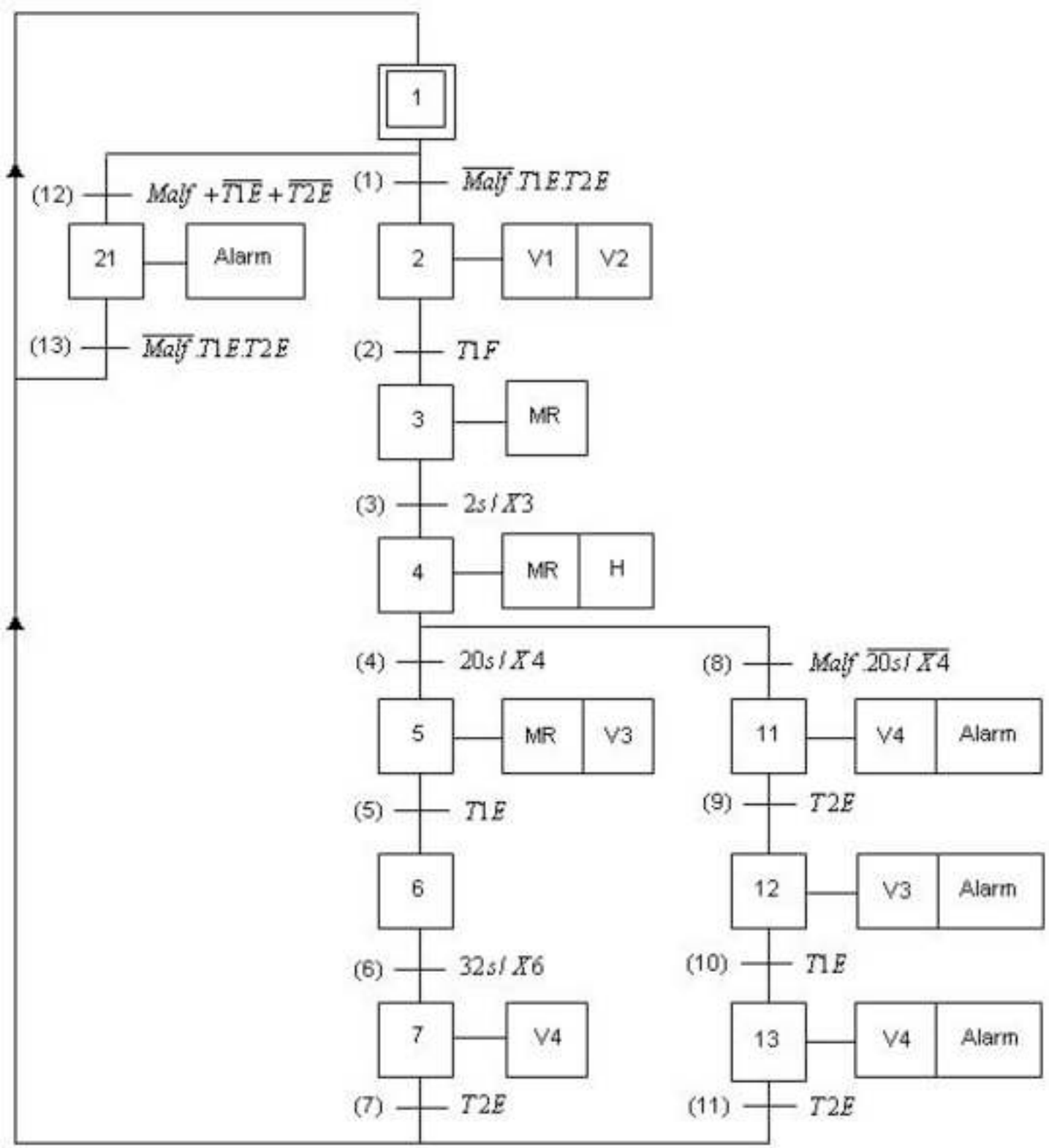

Figure4
Click here to download high resolution image

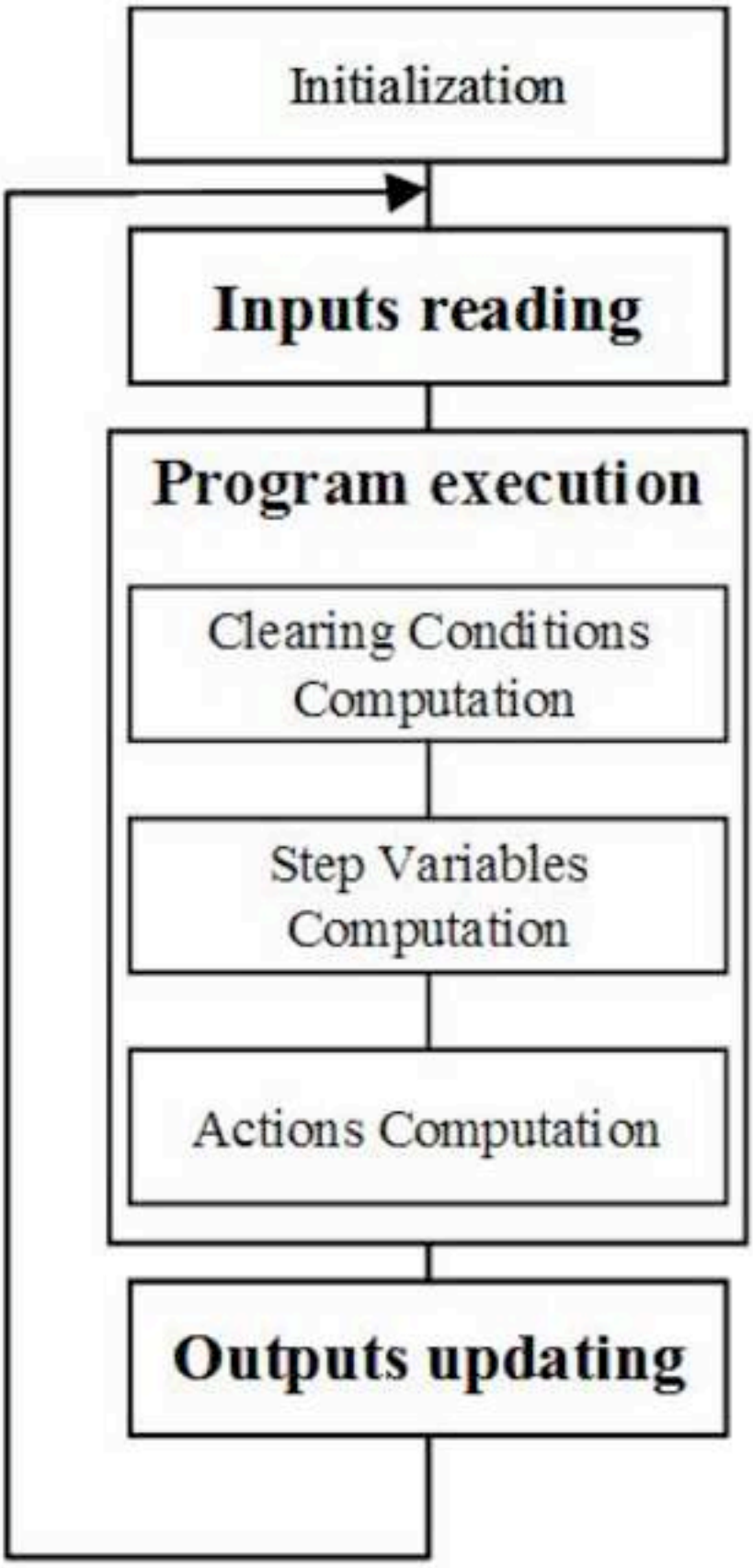

Figure5
Click here to download high resolution image

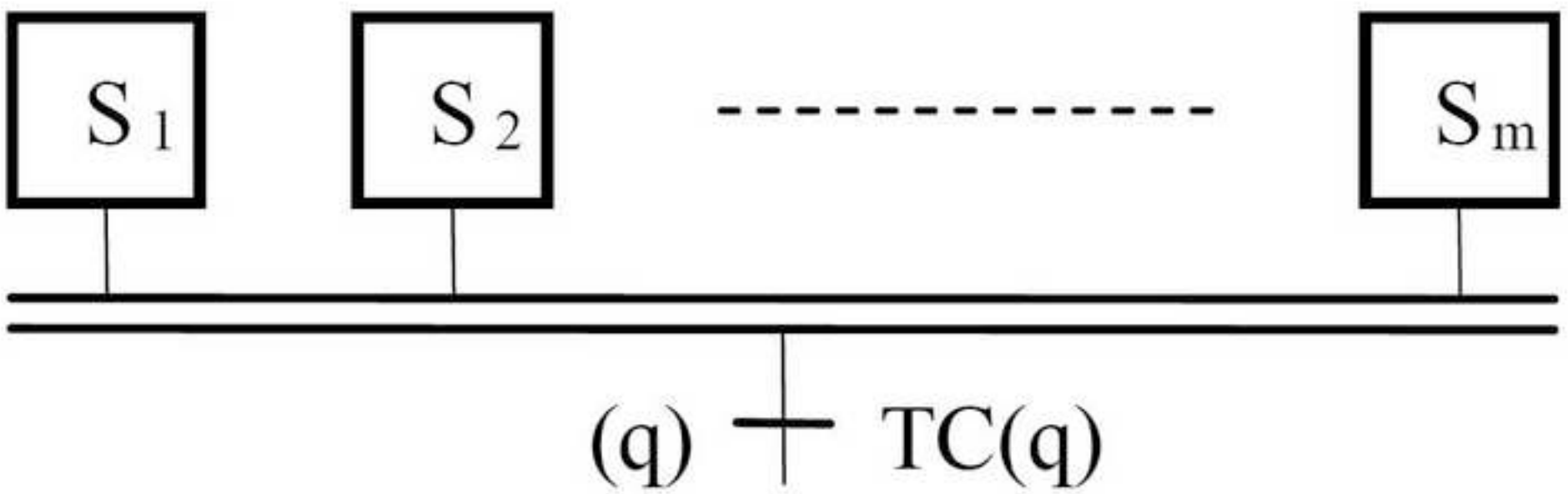

Figure6
Click here to download high resolution image

```
model Tank1
  Modelica.Blocks.Interfaces.RealOutput levelSensor;
  Modelica.StateGraph.Examples.Utilities.inflow inflow1;
  Modelica.StateGraph.Examples.Utilities.outflow outflow1;
  Real level "Tank level in % of max height";
  parameter Real A=1 "ground area of tank in m²";
  parameter Real a=0.2 "area of drain hole in m²";
  parameter Real hmax=1 "max height of tank in m";
  constant Real g=Modelica.Constants.g_n;
  Modelica.StateGraph.Examples.Utilities.inflow inflow2;
equation
  der(level) = (inflow1.Fi + inflow2.Fi - outflow1.Fo)/(hmax*A);
  if outflow1.open then
    outflow1.Fo = sqrt(2*g*hmax*level)*a;
  else
    outflow1.Fo = 0;
  end if;
  levelSensor = level;

end Tank;

connector Modelica.Blocks.Interfaces.RealOutput =
                output RealSignal "'output Real' as connector";

connector Modelica.Blocks.Interfaces.RealSignal
  "Real port (both input/output possible)"
  replaceable type SignalType = Real;

  extends SignalType;

end RealSignal;

connector Modelica.StateGraph.Examples.Utilities.inflow

    import Units = Modelica.SIunits;

  Units.VolumeFlowRate Fi "inflow";
end inflow;

connector Modelica.StateGraph.Examples.Utilities.outflow

    import Units = Modelica.SIunits;

  Units.VolumeFlowRate Fo "outflow";
  Boolean open "valve open";
end outflow;
```

Figure7
Click here to download high resolution image

```
model Tank 2

  Modelica.Blocks.Interfaces.RealOutput levelSensor;
  Modelica.StateGraph.Examples.Utilities.inflow inflow1;
  Modelica.StateGraph.Examples.Utilities.outflow outflow1;
  Real level "Tank level in % of max height";
  parameter Real A=1 "ground area of tank in m²";
  parameter Real a=0.2 "area of drain hole in m²";
  parameter Real hmax=1 "max height of tank in m";
  constant Real g=Modelica.Constants.g_n;
equation
  der(level) = (inflow1.Fi - outflow1.Fo)/(hmax*A);
  if outflow1.open then
    outflow1.Fo = sqrt(2*g*hmax*level)*a;
  else
    outflow1.Fo = 0;
  end if;
  levelSensor = level;

end Tank;

connector Modelica.Blocks.Interfaces.RealOutput =
                output RealSignal "'output Real' as connector";

connector Modelica.Blocks.Interfaces.RealSignal
  "Real port (both input/output possible)"
  replaceable type SignalType = Real;

  extends SignalType;

end RealSignal;

connector Modelica.StateGraph.Examples.Utilities.inflow

    import Units = Modelica.SIunits;

  Units.VolumeFlowRate Fi "inflow";
end inflow;

connector Modelica.StateGraph.Examples.Utilities.outflow

    import Units = Modelica.SIunits;

  Units.VolumeFlowRate Fo "outflow";
  Boolean open "valve open";
end outflow;
```

Figure8

Click here to download high resolution image

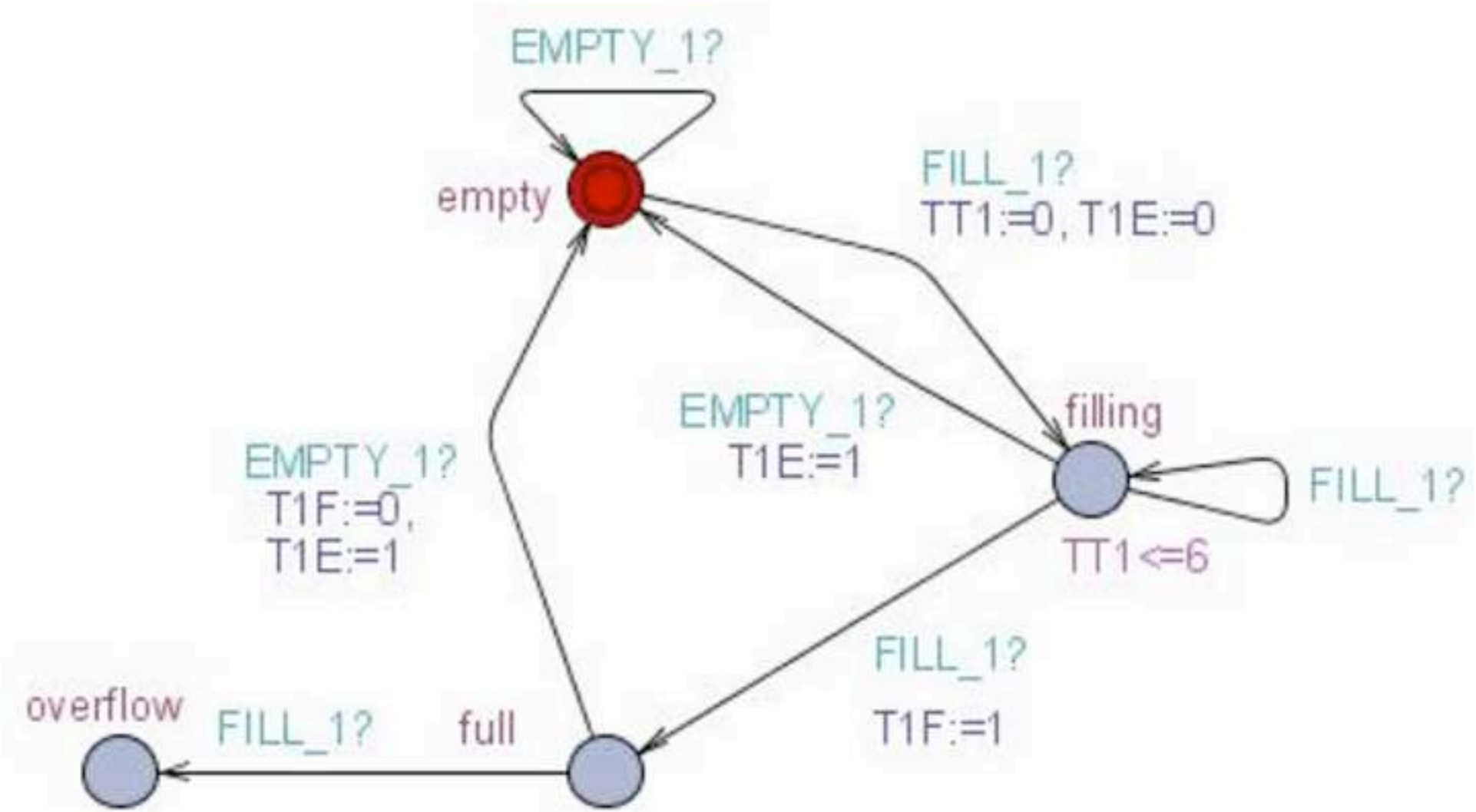

Figure9

Click here to download high resolution image

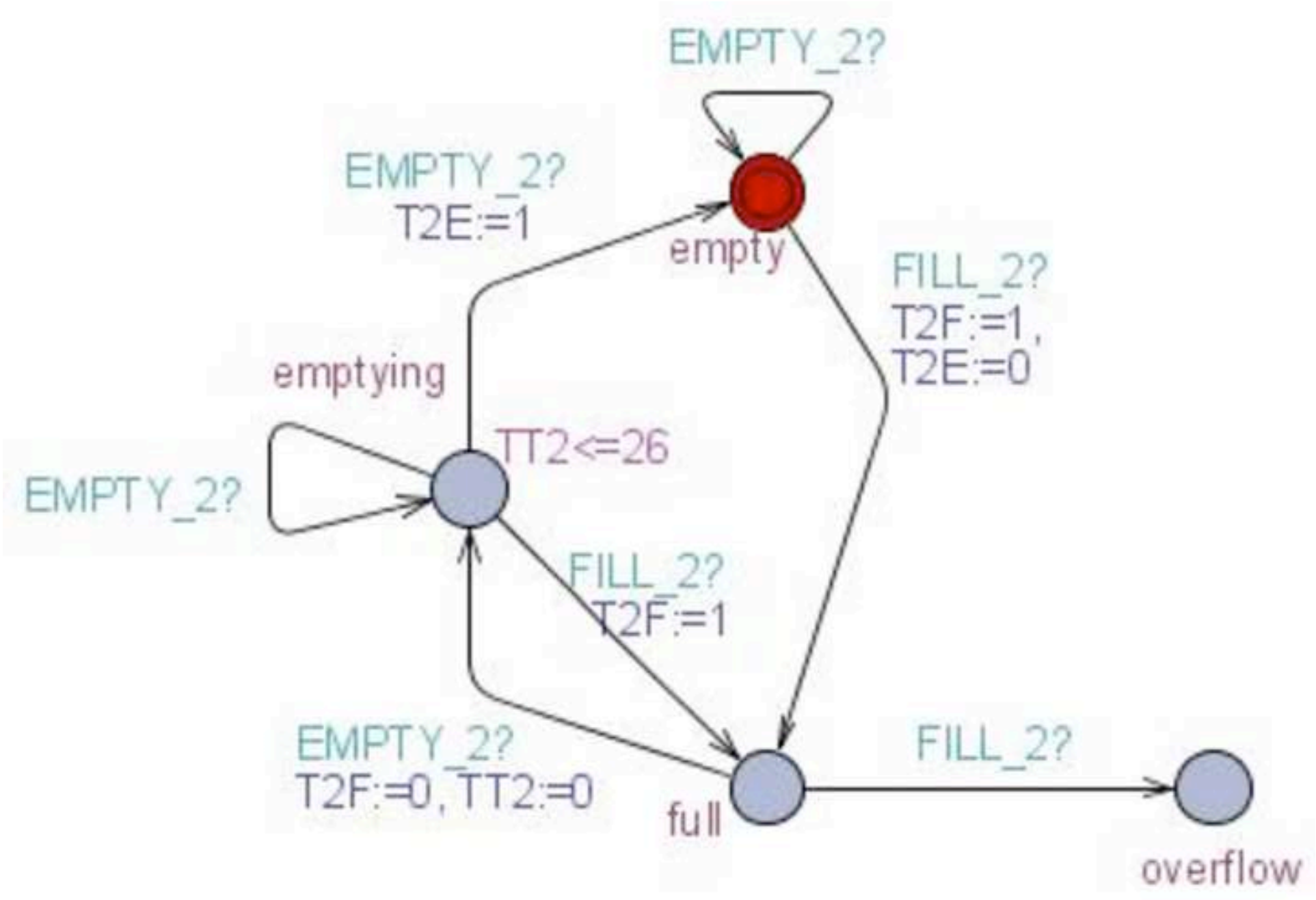

Figure10

Click here to download high resolution image

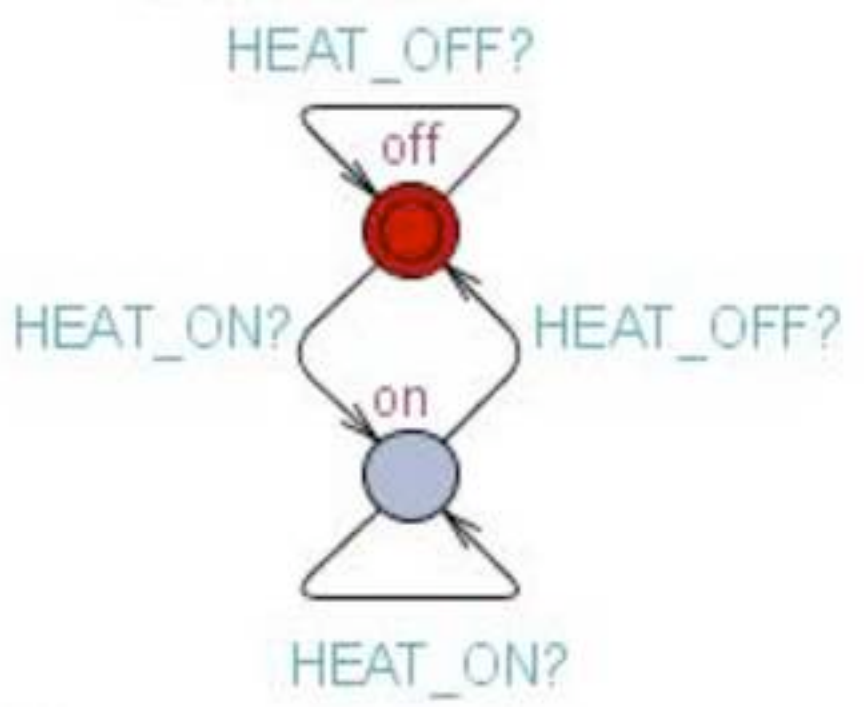

Figure11

Click here to download high resolution image

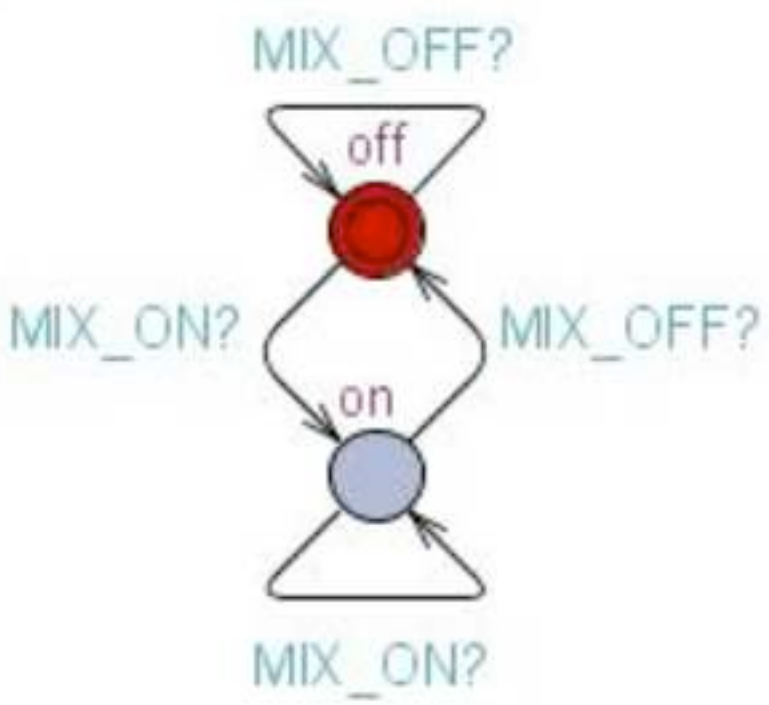

Figure12

Click here to download high resolution image

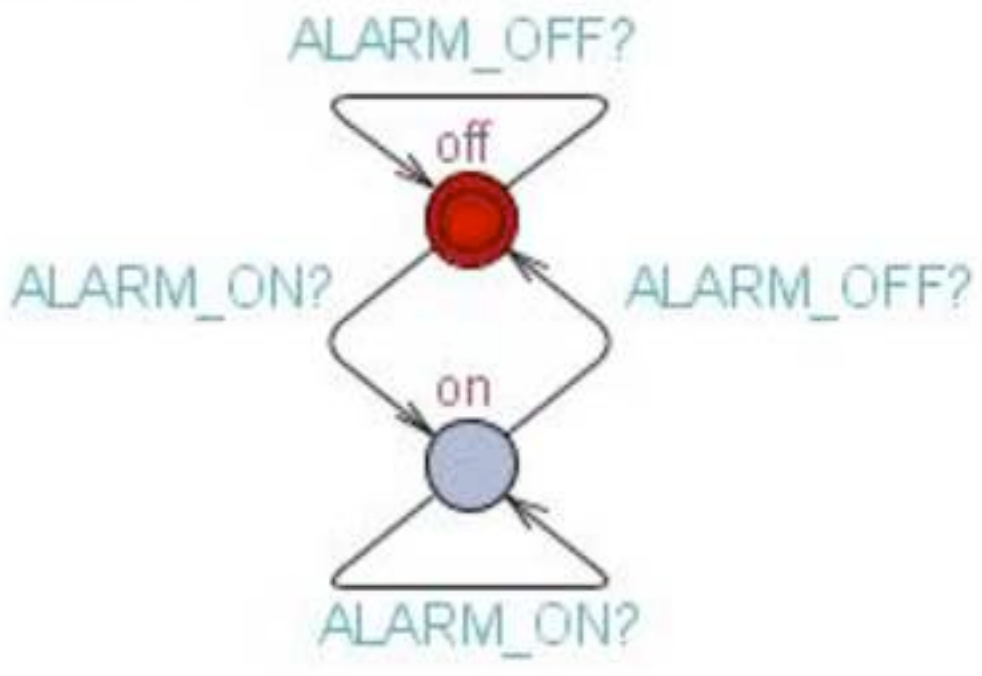

Figure13
Click here to download high resolution image

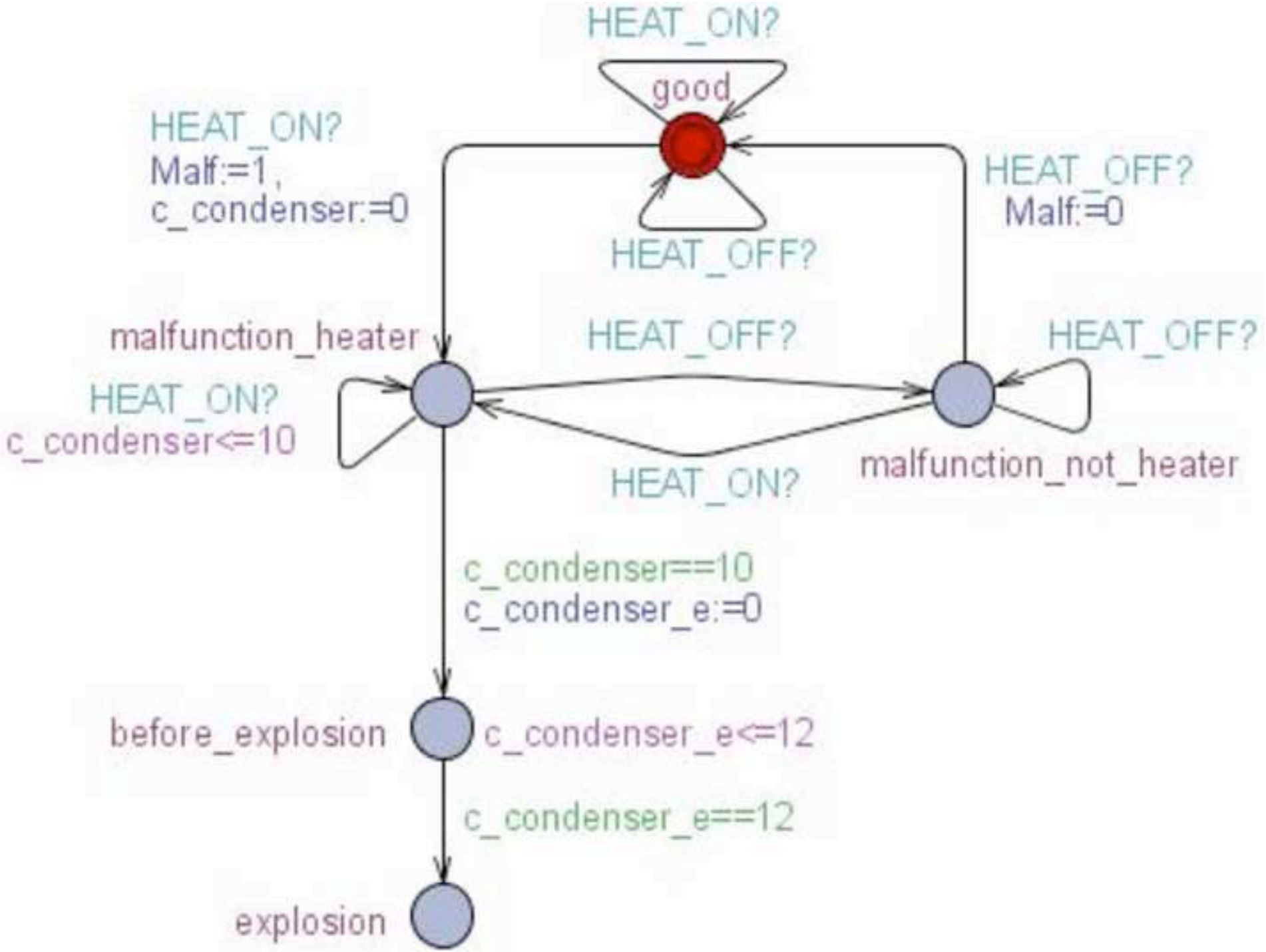

Figure14
Click here to download high resolution image

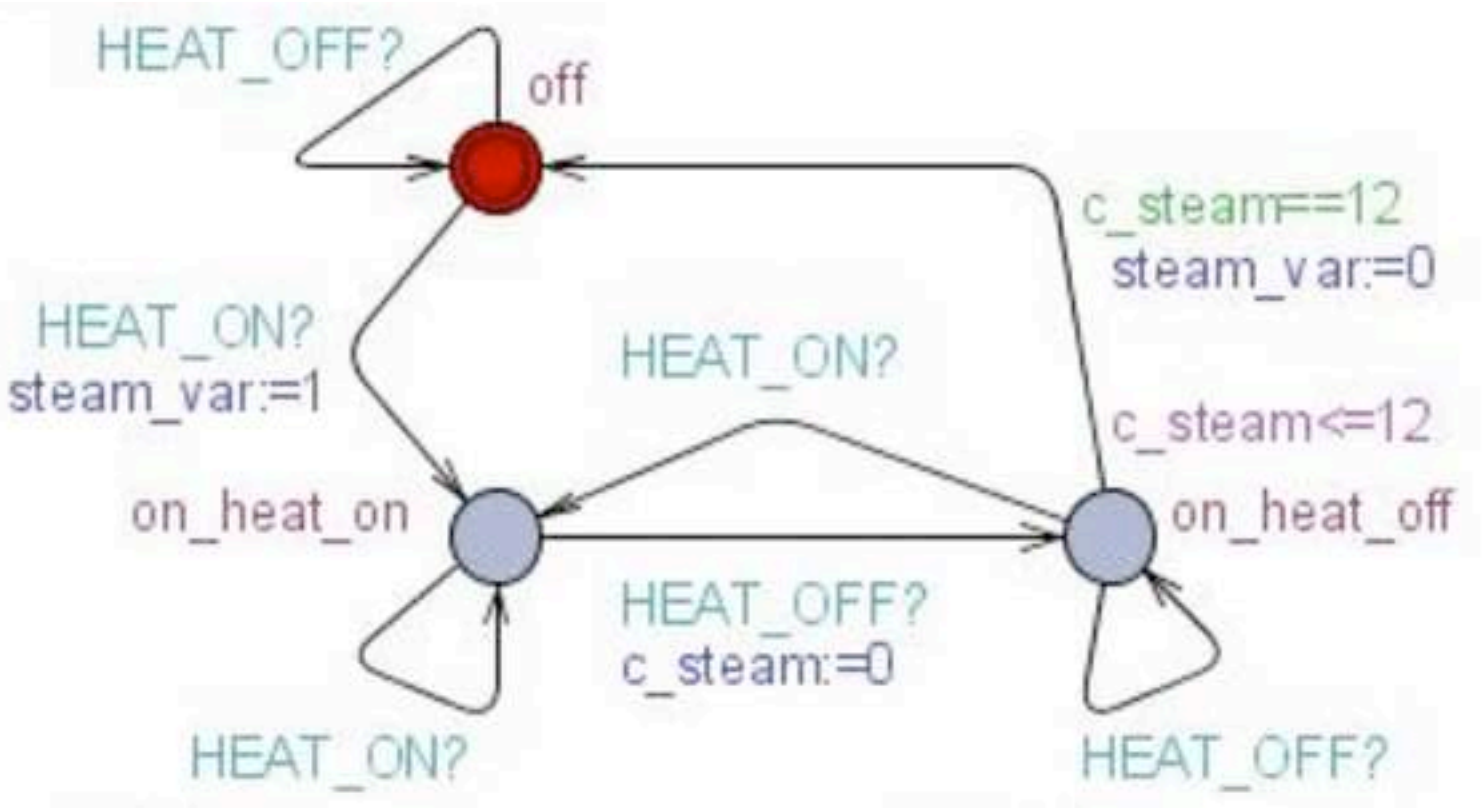

Figure15

Click here to download high resolution image

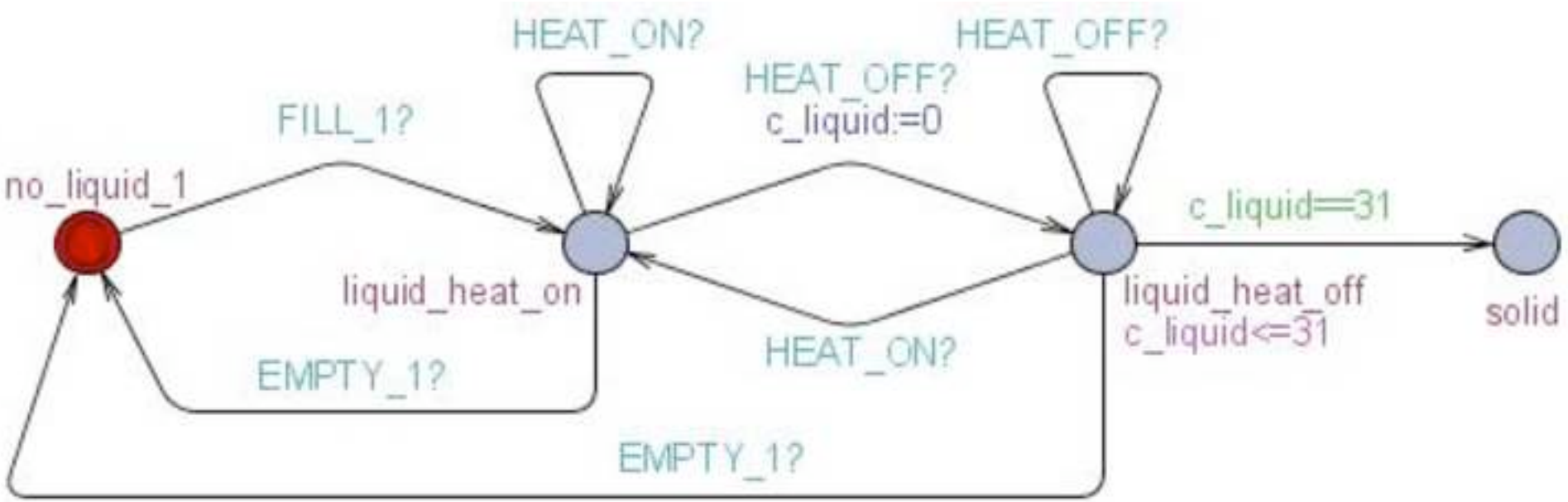

Figure16
Click here to download high resolution image

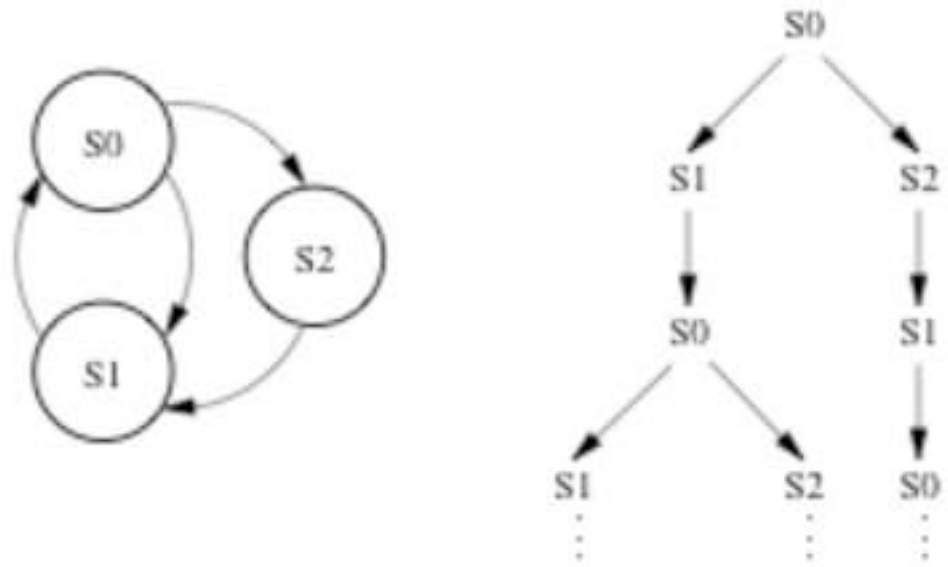

Figure17

Click here to download high resolution image

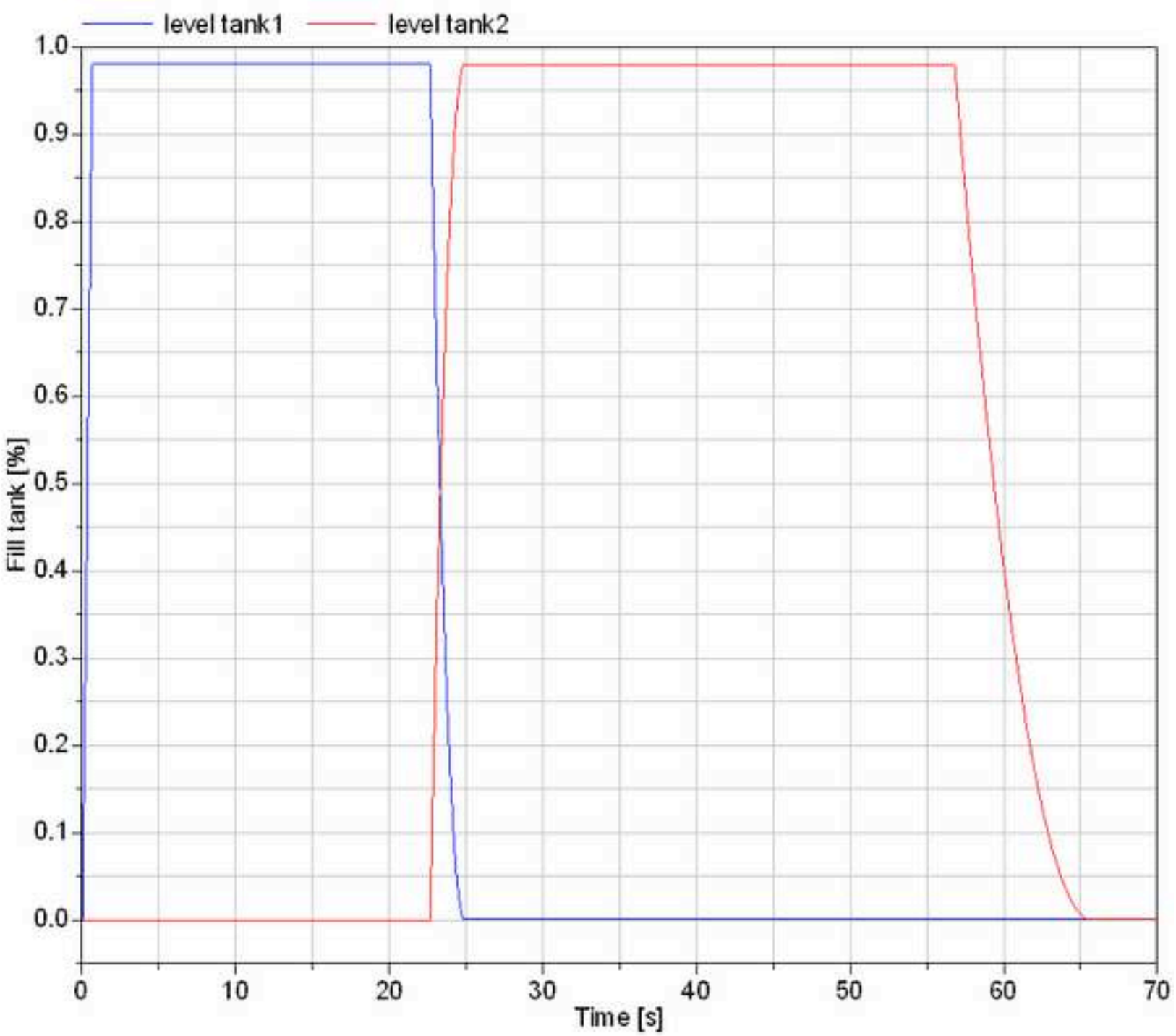

Figure18

Click here to download high resolution image

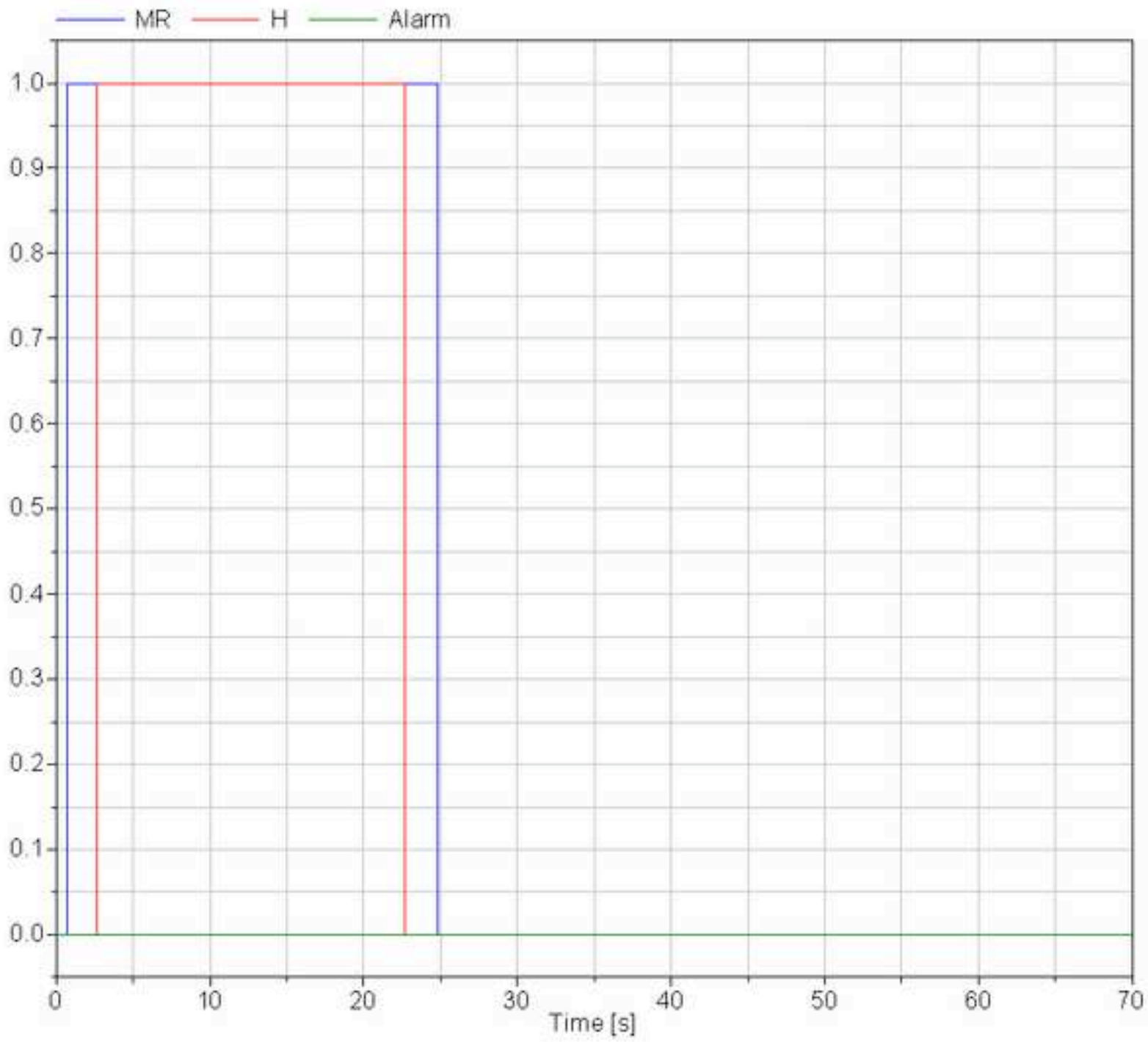

Figure19

Click here to download high resolution image

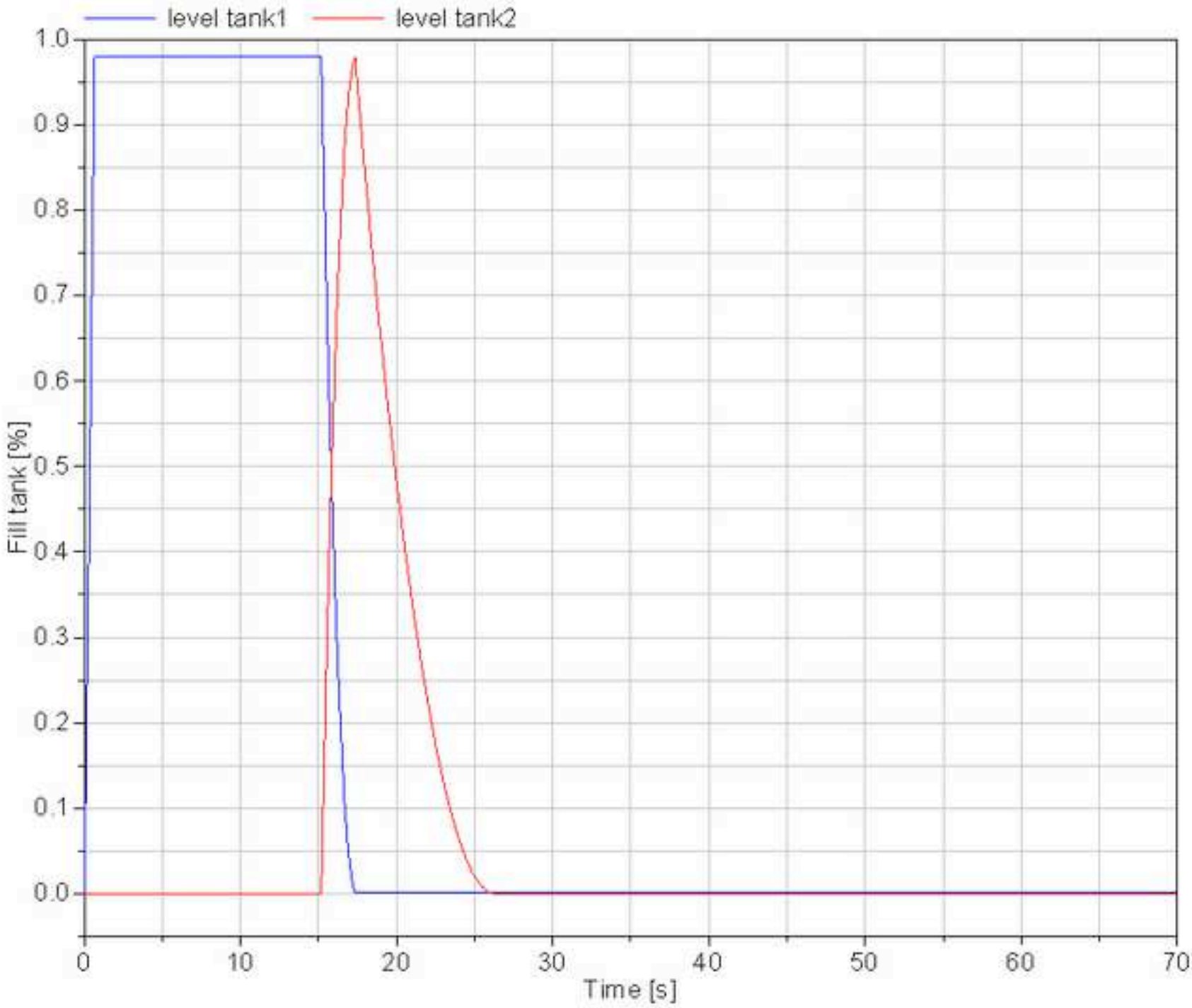

Figure20

Click here to download high resolution image

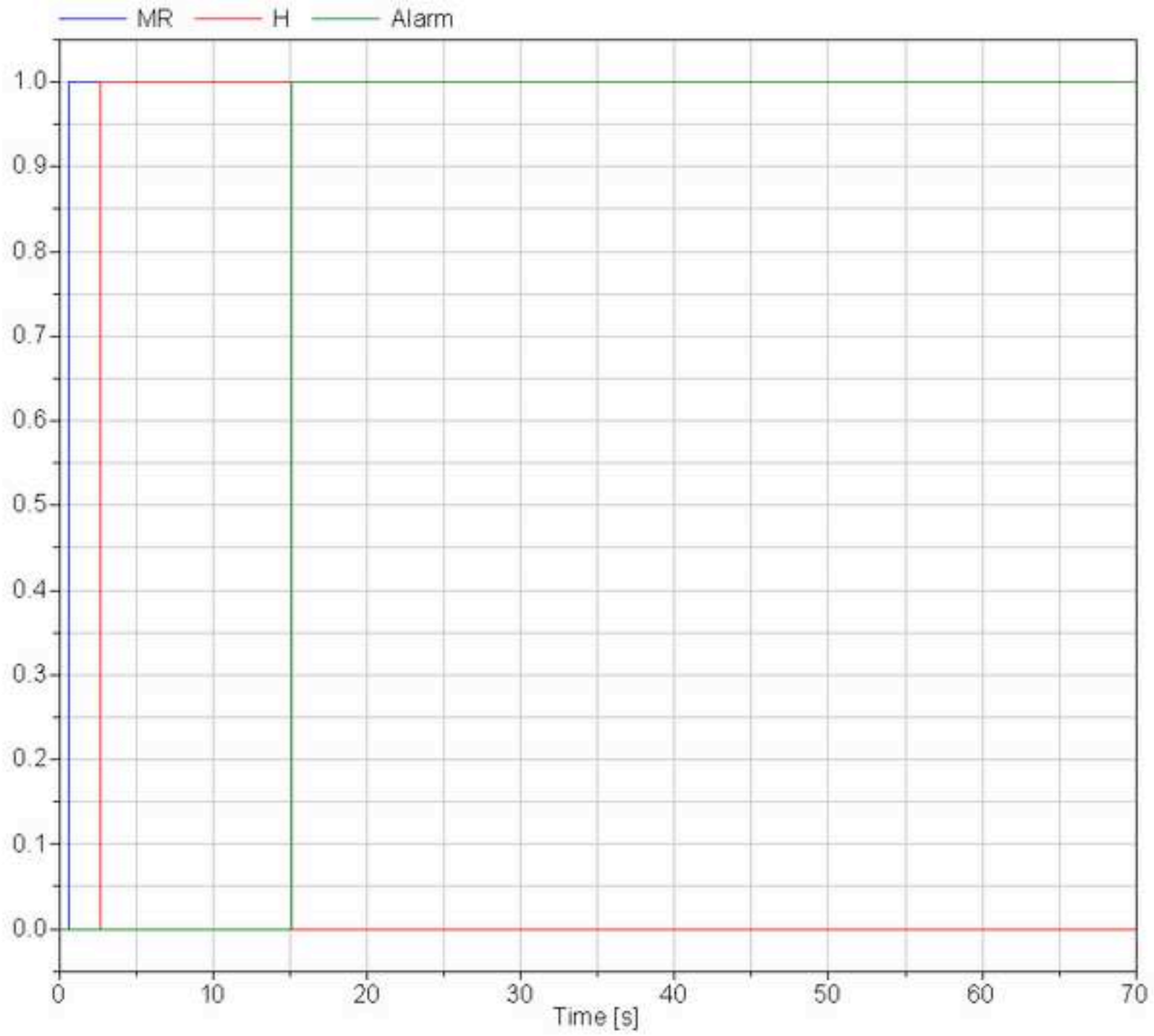

Figure21
Click here to download high resolution image

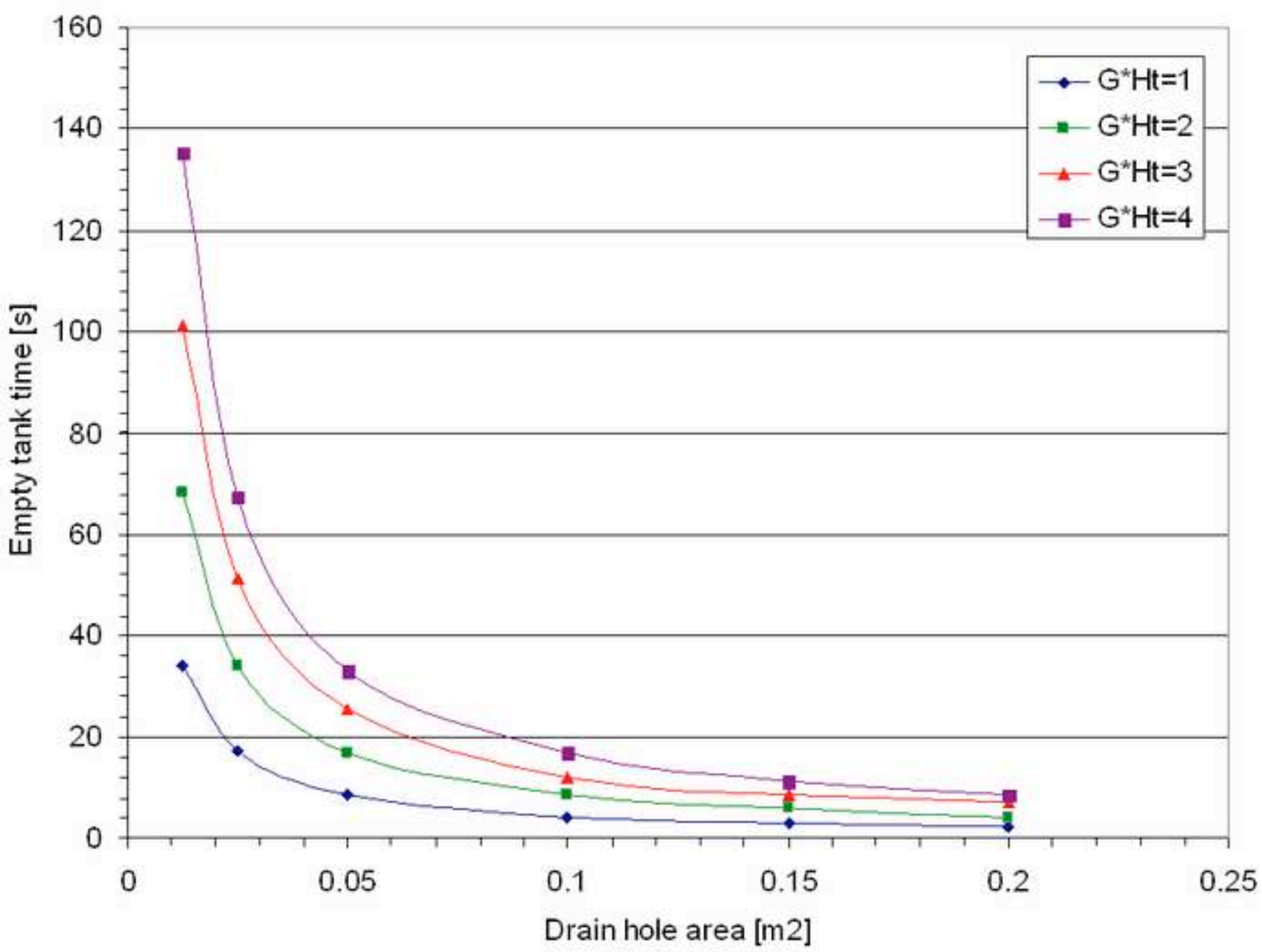

Figure22
Click here to download high resolution image

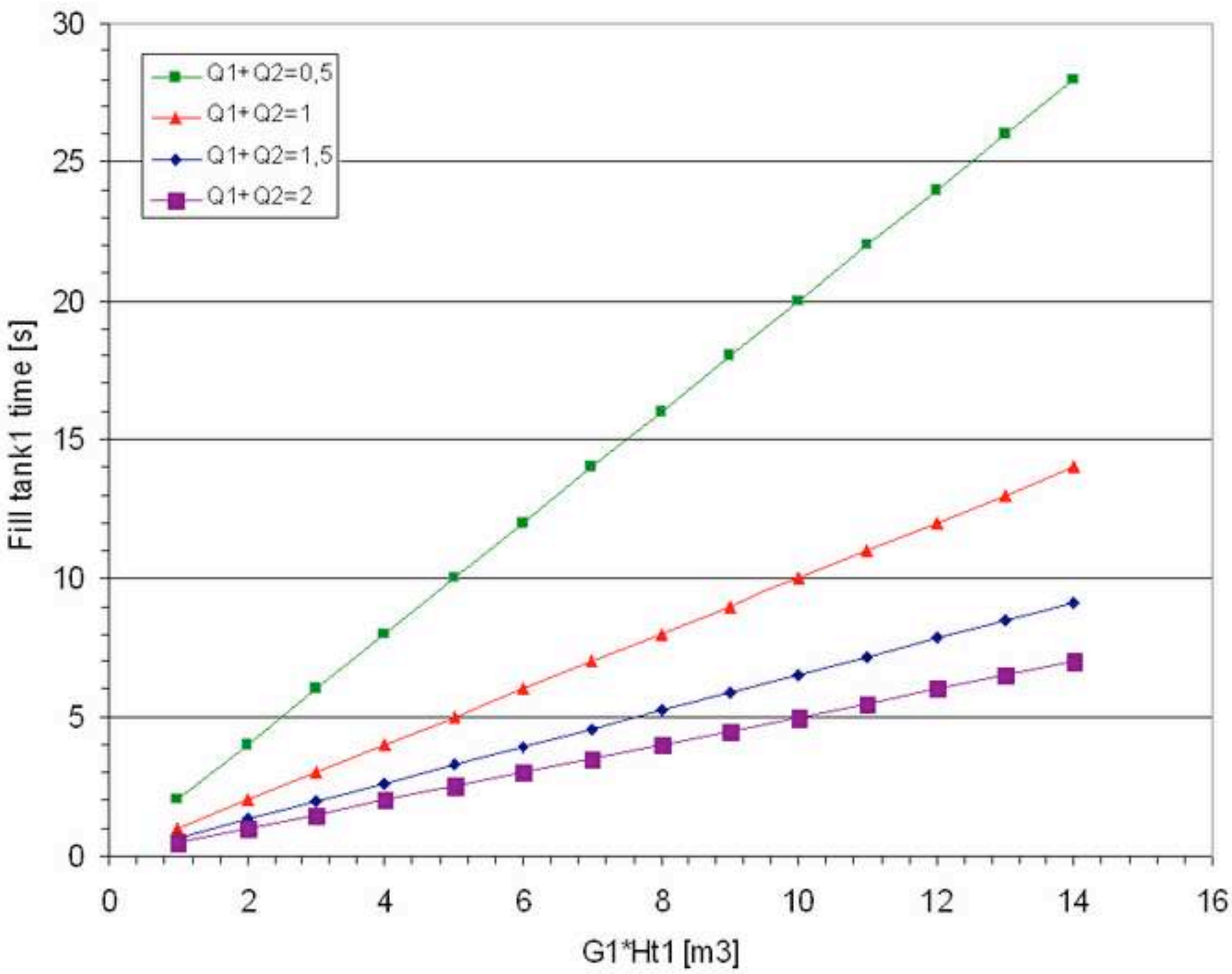

Figure23
Click here to download high resolution image

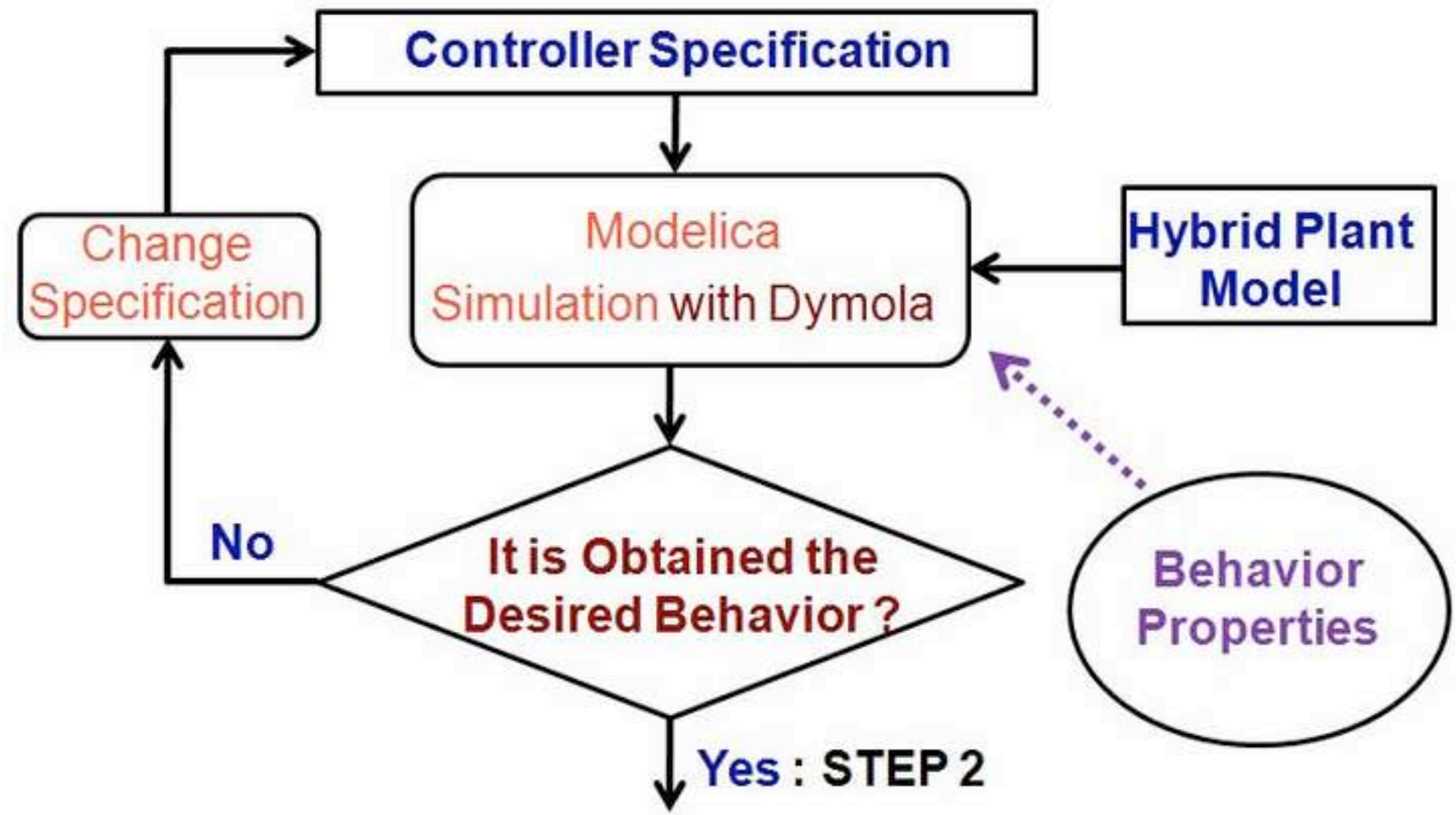

Figure24
Click here to download high resolution image