A Contract-Based Formalism for the Specification of Heterogeneous Systems

Luca Benvenuti[1,2], Alberto Ferrari[2], Leonardo Mangeruca[2],
Emanuele Mazzi[3,2], Roberto Passerone[4,2] and Christos Sofronis[2]

[1]Università di Roma "La Sapienza", Italy
luca.benvenuti@uniroma1.it
[3]Università di Pisa, Italy
emanuele.mazzi@ing.unipi.it

[2]PARADES, Roma, Italy
{leonardo, aferrari, sofronis}@parades.rm.cnr.it
[4]Università di Trento, Italy
roby@disi.unitn.it

Abstract

We present the mathematical formalism and the verification methodology of the contract-based model developed in the framework of the SPEEDS project. SPEEDS aims at developing methods and tools to support "speculative design", a design methodology in which distributed designers develop different aspects of the overall system, in a concurrent but controlled way. Our generic mathematical model of contract supports this style of development. This is achieved by focusing on behaviors, by supporting the notion of "rich component" where functional and non-functional aspects of the system can be considered and combined, by representing rich components via their set of associated contracts, and by formalizing the process of component composition.

1. Introduction

In this paper we describe the semantic structure underlying the metamodel of the SPEEDS integrated project [10], and show an example of application in the field of hybrid systems. SPEEDS is a concerted effort to define the new generation of end-to-end methodologies, processes and supporting tools for safety-critical embedded system design. One of the technical pillars of the SPEEDS approach is the definition of a semantic-based modeling method that supports the construction of complex embedded systems by composing heterogeneous subsystems, and that enables the sound integration of existing and new tools. At the basis of this approach is the definition of a "rich-component" model able to represent functional as well as architectural abstractions, such as timing, safety and other non-functional performance metrics. These different viewpoints can be developed separately in the model, and then integrated and evaluated together in order to derive the most efficient component-based implementation of the system.

To assist this methodology, SPEEDS is developing a new tool-supported process, called *controlled speculative design*, justified by the recent organizational changes and new challenges in industries such as automotive and aerospace. System integrators attempt to recover value from the supply chain by maintaining ownership and focusing on those parts of the design at the core of their competitive advantage, while the rest of the system is centered around standard platforms that could be developed and shared by otherwise competitors. Examples of this trend are AUTOSAR in the automotive industry [6], and the deployment of Integrated Modular Avionics (IMA) in aerospace [4]. A lot of effort has been devoted by the research community during the last years to develop theories and methodologies of component based development to be used in complex supply chains or OEM/supplier organizations such as the ones above. The most familiar case is building systems from component libraries. However, when the structure of the supply chain is highly distributed, this approach is no longer effective, and must be complemented by a precise separation of responsibilities between the different actors of the OEM/supplier chain. In SPEEDS, this is accomplished by establishing formal "contracts" between inter- and intra-company design groups. A contract is a model of a rich component that sets forth the *assumptions* under which the component may be used by its environment, and the corresponding *promises* that are guaranteed under such correct use. OEMs can use the contract mechanism to define and know precisely what a given supplier is responsible for, by assigning a specific goal in the form of a guarantee or promise. The other entities involved in the subsystem may be subject to other constraints that are thus offered to the supplier as assumptions. Assumptions can be used by a supplier to satisfy its own promises. This design style may include both functional and non-functional viewpoints, making the design heterogeneous. This way, designers can more clearly establish and keep track of roles and responsibilities for the development of the system behavior, as well as for its deployment onto optimized architectures.

In this paper, we first review the fundamental semantic concepts of the SPEEDS metamodel, including the notion of contract and how the separation of responsibilities can be captured by a relation of compatibility between contracts (Section 2). Then, we present an example of application of the contract formalism to the control of a hybrid model, which is suitable for heterogeneous discrete and continuous time designs (Section 3), and discuss how to complete its verification (Section 4). Given the limited space, readers interested in a more detailed account and a comparison with related approaches are referred to our previous work [2, 3].

2. Contract model overview

Our model is based on the concept of *component*. A component is a hierarchical entity that represents a unit of design. Components are connected together by sharing and agreeing on the values of certain ports and variables. A component may include both *implementations* and *contracts*. An implementation M is an instantiation of a component and consists of a set P of ports and variables (in the following, for simplicity, we will refer only to ports) and of a set of behaviors, or runs, also denoted by M, which assign a history of "values" to ports. Runs are generic and abstract, since we do not need a predetermined form of behavior for our basic definitions. The particular structure of the runs is defined by specific instances of the model. In Section 3, we show an example of a hybrid model, where runs are composed of continuous flows and discrete jumps. Conversely, an automata based model would represent behaviors as sequences of values or events. Our basic definitions will not vary, and only the way operators are implemented is affected. Because implementations and contracts may refer to different viewpoints, we refer to the components in our model as *heterogeneous rich components* (HRC).

We build the notion of a contract for a component as a pair of assertions, which express its assumptions and promises. An assertion E is modeled as a set of behaviors over ports, precisely as the set of behaviors that satisfy it. An implementation M satisfies an assertion E whenever they are defined over the same set of ports and all the behaviors of M satisfy the assertion, i.e., when $M \subseteq E$.

A contract is an assertion on the behaviors of a component (the promise) subject to certain assumptions. We therefore represent a contract C as a pair (A, G), where A corresponds to the assumption, and G to the promise. An implementation of a component *satisfies a contract* whenever it satisfies its promise, subject to the assumption. Formally, $M \cap A \subseteq G$, where M and C have the same ports. We write $M \models C$ when M satisfies a contract C. There exists a unique maximal (by behavior containment) implementation satisfying a contract C, namely $M_C = G \cup \neg A$, where $\neg A$ denotes the set of all runs that are not runs of A. Clearly, $M \models (A, G)$ if and only if $M \models (A, M_C)$, if and only if $M \subseteq M_C$. Because of this property, we can restrict our attention to contracts of the form $C = (A, M_C)$, which we say are in *canonical form*, without losing expressiveness. The operation of computing the canonical form, i.e., replacing G with $G \cup \neg A$, is well defined, since the maximal implementation is unique, and it is idempotent. Working with canonical forms simplifies the definition of our operators and relations, and provides a unique representation for equivalent contracts.

Intuitively, an implementation can only provide promises on the value of the ports it controls. On ports controlled by the environment, instead, it may only declare assumptions. Therefore, we will distinguish between two kinds of ports: those that are *controlled* and those that are *uncontrolled*. Uncontrollability can be formalized as a notion of receptiveness: for E an assertion, and $P' \subseteq P$ a subset of its ports, E is said to be P'*-receptive* if and only if for all runs σ' restricted to ports belonging to P', there exists a run $\sigma \in E$ such that σ' and σ coincide over P'. In words, E accepts any history offered to the subset P' of its ports. This closely resembles the classical notion of inputs and outputs in programs and HDLs; it is more general, however, as it encompasses not only horizontal compositions within a same layer, but also cross-layer integration such as the integration between application and execution platform performed at deployment. Contracts are therefore enriched with a profile $\pi = (\mathbf{u}, \mathbf{c})$ that partitions its set of ports.

The combination of contracts associated to different components can be obtained through the operation of parallel composition. If $C_1 = (\pi_1, A_1, G_1)$ and $C_2 = (\pi_2, A_2, G_2)$ are contracts (possibly over different sets of ports), the composite must satisfy the guarantees of both, implying an operation of intersection. The situation is more subtle for assumptions. Suppose first that the two contracts have disjoint sets of ports. Intuitively, the assumptions of the composite should be simply the conjunction of the assumptions of each contract, since the environment should satisfy all the assumptions. In general, however, part of the assumptions A_1 will be already satisfied by composing C_1 with C_2, acting as a partial environment for C_1. Therefore, G_2 can contribute to relaxing the assumptions A_1. And vice-versa. The assumption and the promise of the composite contract $C = (\pi, A, G)$ can therefore be computed as follows:

$$A = (A_1 \cap A_2) \cup \neg(G_1 \cap G_2), \quad (1)$$
$$G = G_1 \cap G_2, \quad (2)$$

which is consistent with similar definitions in other contexts [7, 8, 9]. For the profiles, we enforce the property that each port should be controlled by at most one contract. Hence, parallel composition is defined only if the sets of controlled ports of the contracts are disjoint. Likewise, in the composite contract, a port is controlled exactly when it is controlled by one of the component contracts. More formally, parallel composition is defined if and only if $\mathbf{c}_1 \cap \mathbf{c}_2 = \emptyset$, and in that case the profile π is defined by $\mathbf{c} = \mathbf{c}_1 \cup \mathbf{c}_2$ and $\mathbf{u} = (\mathbf{u}_1 \cup \mathbf{u}_2) - (\mathbf{c}_1 \cup \mathbf{c}_2)$.

Typically, C_1 and C_2 may have different ports. In that case, we must extend the behaviors (but not the profiles) to a common set of ports before applying (1) and (2). This can be achieved by an operation of inverse projection. Projection, or elimination, in contracts requires handling assumptions and promises differently, in order to preserve their semantics. For a contract $C = (A, G)$ and a port p, the *elimination of p in C* is given by

$$[C]_p \quad = \quad (\forall p\, A,\ \exists p\, G) \tag{3}$$

where A and G are seen as predicates. Elimination trivially extends to finite sets of ports, denoted by $[C]_P$, where P is the considered set of ports. For inverse elimination in parallel composition, the set of ports P to be considered is the union of the ports P_1 and P_2 of the individual contracts.

Parallel composition can be used to construct complex contracts out of simpler ones, and to combine contracts of different components. Despite having to be satisfied simultaneously, however, multiple viewpoints *associated to the same component* do not generally compose by parallel composition. Take, for instance, a functional viewpoint C_f and an orthogonal timed viewpoint C_t for a component M. Contract C_f specifies allowed data pattern for the environment, and sets forth the corresponding behavioral property that can be guaranteed. For instance, if the environment alternates the values $\mathtt{T}, \mathtt{F}, \mathtt{T}, \ldots$ on port a, then the value carried by port b never exceeds x. Similarly, C_t sets timing requirements and guarantees on meeting deadlines. For example, if the environment provides at least one data per second on port a ($1ds$), then the component can issue at least one data every two seconds ($.5ds$) on port b. Parallel composition fails to capture their combination, because the combined contract must accept environments that satisfy either the functional assumptions, or the timing assumptions, or both. In particular, parallel composition computes assumptions that are too restrictive. We would like, instead, to compute the *conjunction* $\sqcap$ of the contracts, so that if $M \models C_f \sqcap C_t$, then $M \models C_f$ and $M \models C_t$. This can best be achieved by first defining a partial order on contracts, which formalizes a notion of substitutability, or refinement. We say that $C = (\pi, A, G)$ *dominates* $C' = (\pi', A', G')$, written $C \preceq C'$, if and only if $\pi = \pi'$, $A \supseteq A'$ and $G \subseteq G'$. Dominance amounts to relaxing assumptions and reinforcing promises, therefore strengthening the contract. Clearly, if $M \models C$ and $C \preceq C'$, then $M \models C'$.

Given the ordering of contracts, we can compute greatest lower bounds and least upper bounds, which correspond to taking the conjunction and disjunction of contracts, respectively. For contracts $C_1 = (\pi, A_1, G_1)$ and $C_2 = (\pi, A_2, G_2)$ (in canonical form), we have

$$C_1 \sqcap C_2 \quad = \quad (\pi, A_1 \cup A_2, G_1 \cap G_2), \tag{4}$$

$$C_1 \sqcup C_2 \quad = \quad (\pi, A_1 \cap A_2, G_1 \cup G_2). \tag{5}$$

The resulting contracts are in canonical form. Conjunction of contracts amounts to taking the union of the assumptions, as required, and can therefore be used to compute the overall contract for a component starting from the contracts related to multiple viewpoints.

Consistency and Compatibility The notion of receptiveness and the distinction between controlled and uncontrolled ports is at the basis of our relations of consistency and compatibility between contracts. Our first requirement is that an implementations M with profile $\pi = (\mathbf{u}, \mathbf{c})$ be $\mathbf{u}$-receptive, formalizing the fact that an implementation has no control over the values of ports set by the environment. For a contract C, we say that C is *consistent* if G is $\mathbf{u}$-receptive, and *compatible* if A if $\mathbf{c}$-receptive.

The sets A and G are not *required* to be receptive. However, if G is not $\mathbf{u}$-receptive, then the promises constrain the uncontrolled ports of the contract. In particular, the contract admits no receptive implementation. This is against our policy of separation of responsibilities, since we stated that uncontrolled ports should remain entirely under the responsibility of the environment. Corresponding contracts are therefore called *inconsistent*.

The situation is dual for assumptions. If A is not $\mathbf{c}$-receptive, then there exists a sequence of values on the controlled ports that are refused by all acceptable environments. However, by our definition of satisfaction, implementations are allowed to output such sequence. Unreceptiveness, in this case, implies that a hypothetical environment that wished to prevent a violation of the assumptions should actually prevent the behavior altogether, something it cannot do since the port is controlled by the contract. Therefore, unreceptive assumptions denote the existence of an incompatibility internal to the contract, that cannot be avoided by any environment.

The notion of consistency and compatibility can be extended to pairs of contracts. We say that two contracts C_1 and C_2 are *consistent* or *compatible* whenever their parallel composition is consistent or compatible.

3. A hybrid system example

To show how to perform the verification of contract compatibility, we consider a simple hybrid system control problem where we have to control the water level in a cylindric tank (see Figure 1), with height H and section S, equipped with an inlet pipe at the top and an outlet pipe at the bottom. The outlet flow is assumed to be proportional to the water level, i.e.,

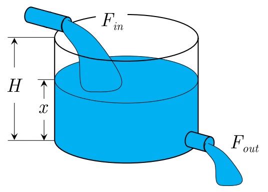

Figure 1. The tank system

$$F_{out}(t) = kx(t) \tag{6}$$

where $x(t)$ denotes the water level in the tank and k is the outlet flow constant. The inlet flow depends on the supply inlet pressure $p(t)$, that is $F_{in}(t) = S_{in}\sqrt{2p(t)/\rho}$ where S_{in} is the inlet pipe cross section and ρ is the liquid density.

The controller scheme considered in this section is shown in Figure 2. The closed loop includes: (i) the water tank; (ii) an actuator consisting of a flow rate valve; (iii) a water level sensor; (iv) a controller that on the basis of sensor readings and reference signal, actuates the valve. We now introduce the contracts of each element of the configuration and verify the compatibility of the connection.

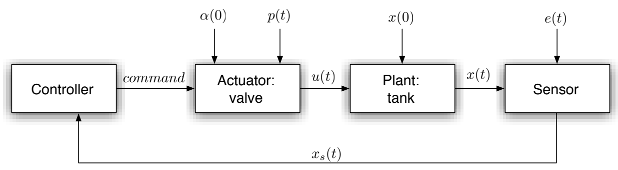

Figure 2. System diagram

The behavior of the tank can be described by a contract $C_{tank} = (A_{tank}, G_{tank})$ which defines all the possible inlet and outlet flow behaviors. The evolution of the output flow, that is the promise G_{tank}, can be described by the hybrid system shown in Figure 3 where the continuous state variable $x(t)$ represents the water level in the tank, the continuous input variable $u(t)$ is equal to $F_{in}(t)/S$, and the continuous output variable $y(t)$ is the outlet flow $F_{out}(t)$. Initially, in location q_1, there is no water overflow and the water level $x(t)$ evolves according to the differential equation associated to the location where $\lambda = k/S$. The system remains in q_1 as long as the water level $x(t)$ is lower than H. Location q_2 describes the water overflow situation in which the water level remains constant and equal to H. The assumptions A_{tank} restrict the range of acceptable input behaviors. In particular, the inlet flow is assumed to be non-negative ($A^d_{tank} = \{u(t) \geq 0\}$) and the initial value of the water level an admissible one ($A^{j,o}_{tank} = \{x(0) \in [0, H]\}$). A one-state hybrid system can be used to capture these two invariants.

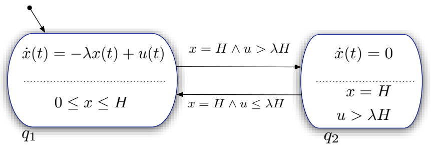
$\dot{x}(t) = -\lambda x(t) + u(t)$
$0 \leq x \leq H$
$x = H \wedge u > \lambda H$
$x = H \wedge u \leq \lambda H$
$\dot{x}(t) = 0$
$x = H$
$u > \lambda H$
q_1
q_2

Figure 3. Hybrid system $\mathcal{H}_t$ for G_{tank}

The inlet flow to the container is controlled by the valve, that takes a position command from a controller. In response to a position command (*open* or *close*), the valve aperture changes linearly in time at rate $1/T$ where $T \in [T^m; T^M]$ s. The inlet flow is proportional to the valve aperture $\alpha(t)$, where $0 \leq \alpha \leq 1$. As a consequence, the input $u(t)$ to the tank is equal to

$$u(t) = \alpha(t)\frac{S_{in}}{S}\sqrt{\frac{2p(t)}{\rho}} = \alpha(t)f\left(p(t)\right), \tag{7}$$

where $p(t)$ is the supply inlet pressure. The assumption of the valve contract is

$$A^d_{valve} = \{p(t) \geq 0\} \times \{T \in [T^m; T^M]\} \times \{\alpha(0) \in [0; 1]\}$$

while the promise G_{valve} is defined on the output $u(t)$ of the valve and can be represented by the hybrid system depicted in Figure 4 where the initial location can be either p_2 or p_4. In location p_1 the valve is closed and the output $u(t)$

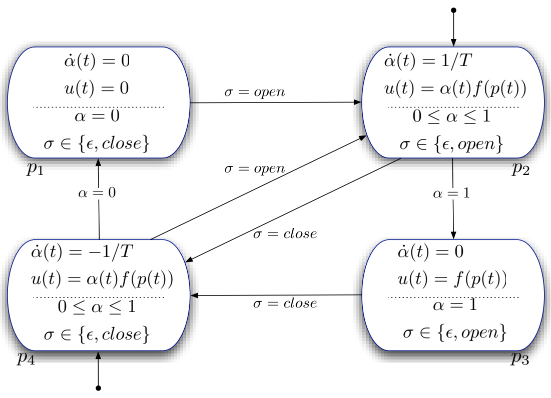

Figure 4. Hybrid system $\mathcal{H}_v$ for G_{valve}

is constantly zero, regardless of the value of the inlet pressure $p(t)$. In locations p_2 and p_4 the valve is opening and closing, respectively. The output $u(t)$ depends on the inlet pressure $p(t)$ and the aperture of the valve $\alpha(t)$. In location

p_3 the valve is open. In this case, the supply outlet flow $u(t)$ depends only on the inlet pressure $p(t)$.

For the sensor, the input is the water level $x(t)$ in the tank, and the output is the measured signal $x_s(t)$ subject to the sensor error $e(t)$. Two invariants are taken as assumptions: the error $e(t)$ is bounded in a given interval ($A^d_{sensor} = \{e(t) \in [-\delta, \delta]\}$); the water level is bounded by the size of the tank ($A^{j,o}_{sensor} = \{x(t) \in [0, H]\}$).

The promise G_{sensor} of the sensor contract is defined simply as the invariant $x_s(t) = x(t) + e(t)$.

The controller takes as input the measured water level $x_s(t)$ provided by the sensor, and outputs the command signal *open* or *close* for the regulation of the valve position. The control law, shown in Figure 5, is based on a hysteresis loop, introduced to reduce the number of actuator commutations and prevent a chattering behavior of the system. When the valve is closed and the water level is decreasing, the controller produces the *open* command only for $x_s(t) \leq l$. Conversely, when the valve is open and the water level is increasing, the controller produces the *close* command only for $x_s(t) \geq h$, with $\delta < l < h < H - \delta$. There are no assumptions A_{contr} on the controller behavior, i.e., any value for $x_s(t)$ is admissible.

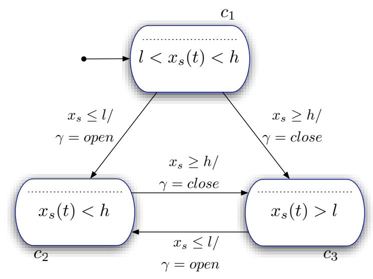

Figure 5. Hybrid system $\mathcal{H}_c$ for G_{contr}

4. Verification

In this section we analyze the compatibility and satisfaction relation. To establish compatibility, we observe that whenever the promises of a component over a certain port are contained in the assumptions of another component for the same port, then the assumptions are discharged. This follows directly from our definition of composition (Equation 1). We will therefore proceed in steps through the cascade composition of Figure 2. Frist, the controller and the valve are compatible, since the valve has no assumptions on the *command* signal. As a second step, consider the composition of the three components controller–valve–tank. The assumptions of the valve ensure through its promise G_{valve}, that $u(t) \geq 0$. Consequently, $G_{valve} \subset A^{j,o}_{tank}$, so that the composition is compatible. Consider now the four-component cascade composition. Also in this case, it is immediate to check that the hybrid system describing the tank behavior is such that $x(t) \in [0, H]$. Therefore, $G_{tank} \subset A^{j,o}_{sensor}$, which concludes compatibility verification of the cascade composition.

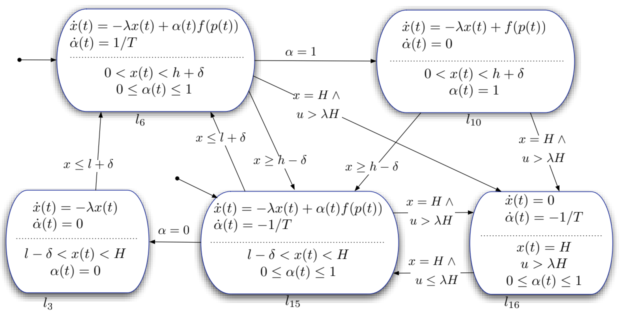

Figure 6. Hybrid system $\tilde{\mathcal{H}}_{tscv}$ for the system

As a final step, consider the composition obtained by closing the loop. Since there are no assumption A_{contr} on $x_s(t)$,

$$G^f = G_{sensor} \subset A_{contr} = A^f$$

and the feedback composition is compatible. The entire system has, however, some residual assumptions on the environment, corresponding to the uncontrolled ports:

$$A^e = A^d_{valve} \times A^d_{tank} \times A^d_{sensor} \tag{8}$$

To compute the global promises, we compose the contracts defining the four blocks of Figure 2 to obtain the contract of the closed–loop system. The result is shown in Figure 6, which has been obtained by applying several reduction techniques, that includes hiding internal signals, eliminating locations with empty invariant conditions, and by computing equivalent locations [3]. Locations l_{10} and l_3 represent the open and closed valve situation, respectively. Locations l_6 and l_{15} reproduce the opening and closing dynamic of the valve. Finally, location l_{16} implements the overflow condition.

To study the satisfaction relation, we define the desired behavior of the closed loop system as a contract C_{des} which defines all the admissible inlet and outlet flow behaviors. The aim of the tank, that is the promise G_{des}, is that of providing an outlet flow $F_{out}(t)$ bounded in a given interval

$$y(t) = F_{out}(t) \in [Y^m; Y^M] = [1; 2]\ m^3/s$$

at least after a settling time $\bar{t} = 10\ s$ from startup when the inlet pressure $p(t)$ is constant and equal to $P = 50000\ Pa$ and the tank is empty at startup. In addition, no water overflow is allowed. Due to the algebraic Equation 6, specifications on the variable $y(t)$ can be expressed in terms of specifications on the state variable $x(t)$.

The assumptions A_{des} on the desired behavior can be stated as $p(t) = P$, $x(0) = 0$, and $\alpha(0) = 0$. An implementation $\mathcal{M}$ of the closed–loop system is described by the hybrid system $\tilde{\mathcal{H}}_{tscv}$ with a particular choice of the physical parameters. As an example, consider the implementation corresponding to the set of parameters: tank - $H = 9\ m$, $S = 19,63\ m^2$, $\lambda = 0,0255\ s^{-1}$, $S_{in} = 0,5\ m^2$, $\rho = 1000\ Kg/m^3$; valve - $T = 2\ s$ sensor - $\delta = 0,05\ m$ controller - $l = 2,2\ m$, $h = 3,8\ m$

To verify whether $\mathcal{M}$ satisfies C_{des}, one has to check if the behavior of the implementation under the assumption A_{des} satisfies the contract promises G_{des}, that is $M \cap A_{des} \subset G_{des}$. For this, an over approximation $\mathcal{R}$ of the infinite–time reachable set of the hybrid system $\tilde{\mathcal{H}}_{tscv}$ is computed using the formal verification tool ARIADNE [5, 1]. More in details, the first requirements on the reached sets after 10 seconds from the startup is obtained by computing the finite-time reachable set of the hybrid model $\tilde{\mathcal{H}}_{tscv}$. Figure 7 shows the evolution of the water level, with initial conditions corresponding to valve closed and empty tank. The reached water level is greater than the required threshold after 10 seconds from the startup. Figure 8 shows

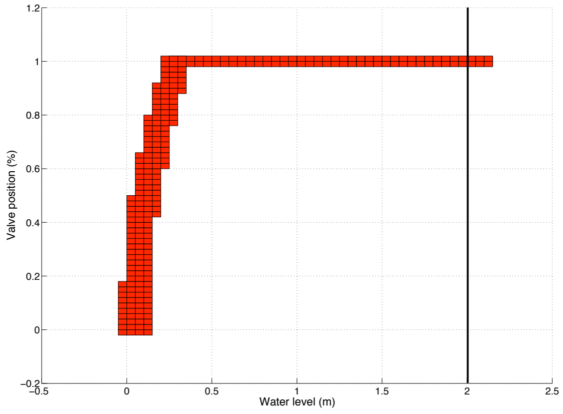

Figure 7. Finite time reachable set

the infinite time reachable set of the hybrid model computed by ARIADNE, to verify if the liquid level is always bounded by the interval $[2; 4]m$. The initial condition is the water level and valve position reached after 10 seconds from startup computed earlier. The regions indicated as B and D are reached when the valve is constantly open and closed, and only the water level $x(t)$ evolves in time (discrete locations l_{10} and l_3). Regions A and C are reached after an open and close controller command, respectively. As the figure makes clear, the water level $x(t)$ is always bounded in the interval $[X^m, X^M] = [2; 4]\ m$. This means that the implementation $\mathcal{M}$ satisfies the constraint on the outlet flow (Equation 6).

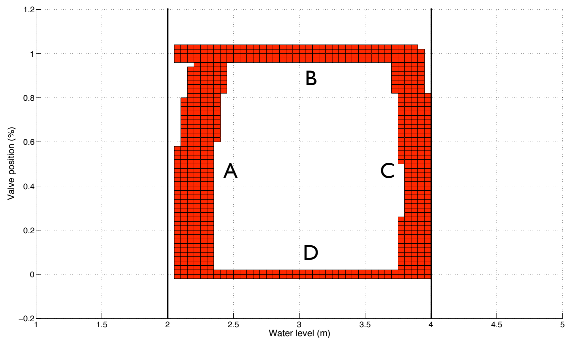

Figure 8. Infinite time reachable set

References

[1] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems*, Kyoto, Japan, July 2006.

[2] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple viewpoint contract-based specification and design. In *Revised Lectures of the Software Technologies Concertation on Formal Methods for Components and Objects (FMCO07)*, Lecture Notes in Computer Science. Springer Verlag, Amsterdam, The Netherlands, 2007. In preparation.

[3] L. Benvenuti, A. Ferrari, E. Mazzi, and A. L. Sangiovanni-Vincentelli. Contract-based design for computation and verification of a closed-loop hybrid system. In *Proceedings of the Hybrid System: Computation and Control (HSCC08)*, St. Louis, MO, April 22–24, 2008.

[4] H. Butz. The Airbus approach to open Integrated Modular Avionics: technology, functions, industrial processes and future development roadmap. In *International Workshop on Aircraft System Technologies, Hamburg*, March 2007.

[5] P. Collins. Continuity and computability of reachable sets. *Theoretical Computer Science*, 341:162–195, 2005.

[6] W. Damm. Embedded system development for automotive applications: trends and challenges. In *Proceedings of the 6^{th} ACM & IEEE International conference on Embedded software (EMSOFT06)*, Seoul, Korea, October 22–25 2006.

[7] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.

[8] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

[9] R. Negulescu. Process spaces. In *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[10] SPEEDS. http://www.speeds.eu.com.