Formally verified architectural patterns of hybrid systems using proof and refinement with Event-B

Guillaume Dupont, Yamine Aït-Ameur, Neeraj Kumar Singh, Marc Pantel

Formally verified architectural patterns of hybrid systems using proof and refinement with Event-B

Guillaume Dupont, Yamine Ait-Ameur, Neeraj Kumar Singh, Marc Pantel

Formally Verified Architectural Patterns of Hybrid Systems Using Proof and Refinement with Event-B[1]

Guillaume Dupont[a], Yamine Ait-Ameur[a], Neeraj Kumar Singh[a], Marc Pantel[a]

[a] *IRIT, ENSEEIHT-INPT*

Abstract

Cyber-Physical Systems (CPSs) are multi-component systems that interact with the real world. Their heterogeneous nature makes them particularly difficult to model and prove.

Our work proposes a framework allowing to design complex hybrid systems based on decomposition using formally verified architectural patterns. It relies on a proof and refinement-based, correct-by-construction approach with Event-B. In particular, the generic model developed in our previous work is used as a base for defining different patterns, for modelling different architectures of systems.

Three architectural patterns are presented, corresponding to simple, centralised and distributed control: one controller controlling one plant, one controller controlling several plants and several controllers controlling several plants.

The progressive development of the proposed patterns and their application to specific hybrid systems allows to prove the required safety properties by introducing invariants, decomposing and balancing the proof effort at every step of the development. An assessment of the proposed architectural patterns is undertaken: different designs of the same case study are used to demonstrate the feasibility, reliability and extensibility of our approach to model and designing different classes of controllers.

1. Introduction

Hybrid systems integrate discrete and continuous behaviours at the same level. The formal development of hybrid systems has attracted the attention of several researchers in the field of formal verification, on the one hand, and control theory, on the other. These two types of research work contributed to define modelling and verification techniques for designing robust hybrid systems by ensuring the relevant properties guaranteeing safe hybrid systems behaviour. The work on formal verification has contributed to the proposal of formal modelling and verification techniques, borrowed from computer science models, enabling to reason on this type of systems and establishing their relevant properties, while control theory is interested in formal modelling based on differential equations to express and guarantee the properties of stability, approximation, linearisation, etc. The challenge is to offer to hybrid systems designers a framework and tools allowing to reason on both discrete and continuous behaviour.

[1] This work was supported by grant ANR-17-CE25-0005 (The DISCONT Project `https://discont.loria.fr`) from the Agence Nationale de la Recherche (ANR).

Email addresses: `guillaume.dupont@enseeiht.fr` (Guillaume Dupont), `yamine@enseeiht.fr` (Yamine Ait-Ameur), `Neeraj.Singh@enseeiht.fr` (Neeraj Kumar Singh), `Marc.Pantel@enseeiht.fr` (Marc Pantel)

URL: `https://www.irit.fr/~Guillaume.Dupont/` (Guillaume Dupont), `http://yamine.perso.enseeiht.fr/` (Yamine Ait-Ameur), `http://singh.perso.enseeiht.fr/` (Neeraj Kumar Singh), `http://pantel.perso.enseeiht.fr/` (Marc Pantel)

The formal verification approaches integrate modelling and verification techniques from both fields of study. An example is the hybrid automation [1] that integrates the notion of state-transition systems with state variables characterised by differential equations. The nature of these equations (linear or non-linear) influences the verification procedure by hybrid model checking. Another example concerns the hybrid programs of [2], which extend dynamic logic with the possibility of handling state variables also characterised by differential equations. The definition of differential invariants and the use of equation solvers are introduced. These two examples clearly illustrate the successful integration of these two fields.

In addition, the same techniques are also interesting for modelling commonly used controllers, such as PIDs [3], the techniques of resolution and discretisation of differential equations [4]. They rely on the definition of generic models that can be instantiated for the design and verification of specific hybrid systems.

There exist several approaches integrating continuous features in a pre-existing formal method. For instance, Hybrid Event-B [5] extends Event-B's language, semantics and method as to define continuous behaviour (so-called *pliant* events), closely woven with discrete events (*mode* events). Continuous dynamics are modelled by functions of time, and can be expressed using differential equations.

Other examples of such approach include Hybrid CSP [6], an hybridation of Hoare's Communicating Sequential Processes [7], and Continuous Action System [8], a continuous extension of Back's Action Systems [9].

Note that the presented approaches rely on modelling systems and performing proofs on models; so-called *bottom-up* approaches consist in working directly with the controller's code (e.g. written in C) and to annotate it in order to incorporate any relevant information on the continuous dynamics of the system. From these annotations (in ACSL or Frama-C for instance), proof obligation are extracted, and discharged using Coq [10] or PVS [11].

Our previous work

In the same vein, our work is interested in the definition of generic and reusable models to design and verify hybrid systems. Our approach is based on refinement and proof with the Event-B method, and on a generic model.

Such generic model has been defined [12] and improved [13], and has already been used on various case studies, including an automatically braking car [12] and a signalised left-turn assist system [14], as well as in the context of railway systems [15].

In our later work, we took interest in the architectural features of hybrid and cyber-physical systems. In particular, we studied the case of one controller controlling several plants with a case study based on liquid tanks [16]. This case study is expanded to build a model featuring several controllers each controlling one plant [17] (so-called many-to-many). We use to derive common features and rules for such systems.

In this paper, we put together the case studies formerly developed and expand on them to build proper reusable *formal design patterns*, under the form of generic models. In particular, we investigate the proof aspect of these patterns, and provide help in discharging them, based on the form of the model, guaranteed by the use of the patterns. Such patterns put together offer the capability to a CPS designer to build any specific system via a series of pattern applications, starting with the generic model. Reusability of models and proofs is central in this approach.

Last, we also propose a discussion on the *feasibility* of the application of such patterns, as they introduce constraints that influence the possible behaviours of the system.

Component-based design of hybrid systems

Hybrid systems with a high level of complexity are substantially harder to model and verify. A common technique to overcome this problem is to decompose the system in several simpler components, for which verification is easier, or, dually, to use common basic components and to compose them together to build a complex system. Architecture, in terms of how the components of a hybrid system interact, is thus an essential phase of hybrid system design.

When performing composition/decomposition, care must be taken to ensure each component's behaviour is compatible with the behaviour of other components. Additionally, the specific behaviour of each sub-component shall guarantee the safety requirements expressed on the whole system expressed in the form of a global invariant. This design process has been set up by different approaches.

To that extent, the work of Lunel et al. [18, 19] extends the hybrid program modelling approach [2] with two composition operators, that permit a new system to be built from two existing systems. The authors have also extended the proving system, enabling proof decomposition: any proof on the global system can be decomposed into simpler proofs, under the condition that the operators are correctly used.

More generally, refinement operations are the support of system composition and decomposition steps. It allows to focus on a system at an abstract level and refine to a more complex system with the same properties, effectively balancing the verification effort. This technique is commonly used in approaches such as Event-B and CSP, are thus made available in their hybrid extensions (Hybrid Event-B [20, 21] and Hybrid CSP [22]). Hybrid CSP is successfully used in the development of the Chinese lunar lander [23] and in the verification of level 3 of the Chinese Train Control System (CTCS-3) [24].

In the same vein, differential refinement logic [25] also enables differential dynamic logic (d$\mathcal{L}$) to refine a system and decompose it, in order to ease the proving process and improve the re-usability of its components. It introduces the notion of differential invariants to prove the correctness of this refinement. This approach has been applied to model hybrid systems like car control system [26], train control system [27], flight collision avoidance system [28] and mobile robots and surgical robots [29].

Banach extended the Event-B modelling language to Hybrid Event-B [20, 21] by other constructs that enable the modelling of hybrid systems. In particular, it proposes the possibility of defining a whole system based on the modelling of several components, specified and interacting through their *interfaces* connecting the components with one another. The defined approach provides with a number of assumptions and requirements, under the form of proof obligations, to establish the correctness of the system. The defined model leads to a set of proof obligations that ought to be discharged. Several applications have been reported in [30, 31].

Contribution of this paper

In this paper, our objective is to show how our Event-B based generic model for hybrid systems design can be used to define generic hybrid systems patterns defining different architectures of such systems. We specify three Event-B based models for three architectural patterns addressing different types of control:

- control of a single plant, where one controller handles one plant;
- centralised control of multiple plants, where one controller handles several plants;
- distributed control of multiple plants, where several controllers each handle one plant and satisfy global properties.

All these patterns ensure safety properties through the guarantee of the defined invariants. Throughout this article we will use the same case study, and show how the different architectural patterns are instantiated to obtain a concrete model associated with this case study.

Organisation of this paper

Our paper is structured as follows. Section 2 recalls the basics of the Event-B modelling language. Section 3 presents the mathematical extensions, in the form of formal theories, of the Event-B modelling language to integrate both discrete and continuous modelling in the same setting. It also includes the basic concepts of the hybrid systems generic model we defined. Section 4 describes a case study, which is used throughout this article for modelling different classes of controllers using the three architectural patterns sketched above. The next sections are devoted to the presentation of these architectural patterns. Sections 5, 6 and 7 present the *single-to-single*, *single-to-many* and *many-to-many* architectural patterns respectively. Each pattern description outlines the Event-B formal development, the corresponding proofs and its application to the proposed case study. Finally, Section 8 concludes the paper and outlines some possible future work.

2. Event-B and its Extension Using Theories

Event-B [32] is a state-based modelling language primarily established on set theory and first order logic. It enables the *correct-by-construction* approach to the design of a complex and large system.

A set of events allows for state changes[2] (see Table 1). For each model, a set of proof obligations (see Tables 2a and 2b) is generated automatically. Event-B is equipped with powerful proof systems, tactics, including a set of proof rules for formal reasoning. The system model consists of an abstract model that leads to the final concrete model, where each refinement introduces additional features and required safety properties that adopt different design decision.

[2]**Notation.** The superscripts ^{A} and ^{C} denote abstract and concrete features.

Context	Machine	Refinement
CONTEXT Ctx **SETS** s **CONSTANTS** c **AXIOMS** A **THEOREMS** T_{ctx} **END**	**MACHINE** M^A **SEES** Ctx **VARIABLES** x^A **INVARIANTS** $I^A(x^A)$ **THEOREMS** $T_{mch}(x^A)$ **VARIANT** $V(x^A)$ **EVENTS** **EVENT** evt^A **ANY** α^A **WHERE** $G^A(x^A, \alpha^A)$ **THEN** $x^A :\mid BAP^A(\alpha^A, x^A, x^{A'})$ **END** ...	**MACHINE** M^C **REFINES** M^A **VARIABLES** x^C **INVARIANTS** $J(x^A, x^C) \wedge I^C(x^C)$ ... **EVENTS** **EVENT** evt^C **REFINES** evt^A **ANY** α^C **WHERE** $G^C(x^C, \alpha^C)$ **WITH** $x^{A\prime}, \alpha^A : W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C'})$ **THEN** $x^C :\mid BAP^C(\alpha^C, x^C, x^{C'})$ **END** ...
(a)	(b)	(c)

Table 1: Model structure

Context (Table 1.a). `Context` is a modelling component for describing the static properties of a system. It includes the definitions, axioms and theorems needed to describe the required concepts using elementary components such as *Carrier sets* s, *constants* c, *axioms* A and *theorems* T_{ctx}.

Machines (Table 1.b). `Machine` is another modelling component to describe model behaviour as a transition system using core elementary components such as *variables* x, *invariants* $I(x)$, *theorems* $T_{mch}(x)$, *variants* $V(x)$ and *events* evt. Each event is guarded with predicate and is used to update state variables using Before-After Predicates (BAP) relating current (x) and next (x') values of state variables. It defines a guarded transition.

(1) Theorems	$A \Rightarrow T_{ctx}$ $A \wedge I^A(x^A) \Rightarrow T_{mac}(x^A)$
(2) Invariant preservation (INV)	$A \wedge I_A(x^A) \wedge G_A(x^A, \alpha^A)$ $\wedge BAP^A(x^A, \alpha^A, x^{A'})$ $\Rightarrow I^A(x^{A\prime})$
(3) Event feasibility (FIS)	$A \wedge I_A(x^A) \wedge G^A(x^A, \alpha^A)$ $\Rightarrow \exists x^{A\prime} \cdot BAP^A(x^A, \alpha^A, x^{A\prime})$
(4) Variant progress	$A \wedge I^A(x^A) \wedge G^A(x^A, \alpha^A)$ $\wedge BAP^A(x^A, \alpha^A, x^{A\prime})$ $\Rightarrow\ V(x^{A\prime}) < V(x^A)$

(a) Machine Proof obligations

(5) Event Simulation (SIM)	$A \wedge I^A(x^A) \wedge J(x^A, x^C)$ $\wedge G^C(x^C, \alpha^C)$ $\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$ $\wedge \boldsymbol{BAP^C(x^C, \alpha^C, x^{C'})}$ $\boldsymbol{\Rightarrow BAP^A(x_A, \alpha^A, x^{A'})}$
(6) Guard Strengthening (GS)	$A \wedge I^A(x^A) \wedge J(x^A, x^C)$ $\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$ $\wedge \boldsymbol{G^C(x^C, \alpha^C) \Rightarrow G_A(x^A, \alpha^A)}$
(7) Invariant preservation (INV)	$A \wedge I^A(x^A)$ $\wedge G^C(x^C, \alpha^C)$ $\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$ $\wedge BAP^C(x^C, \alpha^C, x^{C'})$ $\wedge \boldsymbol{J(x^A, x^C) \Rightarrow J(x^{A'}, x^{C'})}$

(b) Refinement Proof obligations

Table 2: Proof Obligations

Refinements (Table 1.c). Refinement is a process of introducing different characteristics of systems related to functionality, safety and reachability at different abstraction levels. In particular, it decomposes the *machine*, a state-transition system, into a less abstract one, with more design decisions resolved (refined states and events) moving from an abstract level to a less abstract one (simulation relationship). A set of new gluing invariants $J(x^A, x^C)$ relating to abstract and concrete variables always ensures the correctness of the system and the preservation of the property.

Proof Obligations (POs) and property verification. Tables 2a and 2b show a set of proof obligations for the machine and its refined model. Note that these POs are generated automatically and guarantee Event-B model consistency, including refinements. In order to establish the correctness of the defined model, all the generated POs must be proven.

Rodin. It is an Eclipse based IDE for Event-B project management, model edition, refinement and proof, automatic PO generation, model checking, model animation and code generation. It is equipped with standard provers, including support for external provers such as SMT solvers. A plug-in [33] is also available to support the development of mathematical theories.

More details on the Event-B method and Rodin are available in [32] and [34] respectively.

2.1. Extensions with Mathematical Theories

Event-B's expression language is based on first-order logic and set theory, which is mathematically low level. This particularity makes it very expressive, but higher-level constructs are often hard to express and poorly reusable.

```
THEORY Th
IMPORT Th1, ...
TYPE PARAMETERS E, F, ...
DATATYPES
  Type1(E, ...)
  constructors cstr1(p1: T1, ...), ...
OPERATORS
  Op1 <nature> (p1: T1, ...)
    well-definedness WD(p1,...)
    direct definition D1
```

```
AXIOMATIC DEFINITIONS
  TYPES A1, ...
  OPERATORS
    AOp2 <nature> (p1: T1, ...): Tr
      well-definedness WD(p1,...)
  AXIOMS A1, ...
THEOREMS T1, ...
PROOF RULES
  ...
END
```

Listing 1: Elements of Event-B Theory Syntax

To address this issue, an extension has been proposed [35] to define a new type of component, the *theories*. The syntax for theories is detailed in Listing 1. A theory allows to define type-generic data-types with various constructors (`DATATYPE` clause), together with operators, either directly or axiomatically defined. Abstract types may also be defined, similarly to carrier sets in contexts. Types and operators defined in a theory can be used seemlessly in models.

Operators are associated to well-definedness (WD) conditions, that are predicates that characterise when the operator is correctly used (e.g. a/b is well-defined if and only if $b \neq 0$). Upon using an operator with a WD condition in a model, a WD proof obligation is automatically generated.

Last, a theory can define a number of axioms and theorems, and even *proof rules* (specific rewriting and inference rules), that generally encode properties on the operators and types of the theory. These properties may be referenced during the proving process.

3. A Generic Event-B Model for Hybrid Systems

Although Event-B is adapted to the modelling of systems in general, it lacks several features to be able to model *hybrid* systems. Fortunately, Event-B's expression language is based on first-order logic and set theory, which permits to express required continuous features for modelling hybrid behaviour.

In this section, we present a framework for the formal design of hybrid system. This presentation is based on our previous work [13, 36], where we have defined a generic framework for formal hybrid system modelling in Event-B. This generic model defines the core modelling concepts for designing controller-plant loop hybrid systems. It formalises hybrid automata in Event-B, and brings to classical Event-B an implementation of Hybrid Event-B concepts [5].

Note that this section is a summary of these contributions, which are more detailed in the aforementioned publications.

We first introduce a set of continuous features, aimed at modelling continuous behaviour in Event-B models. Second, these features are used to define a generic model, formalising an abstract and generic model of controller-plant loop hybrid systems. The derivation of specific hybrid system models from this model is achieved by refinement.

Note that our intention with the framework is to build it on *core* Event-B, which means we do not change the semantics or the language of Event-B; we simply use *theories* to gather and factorise higher level constructs. Our approach can be seen as *low-level modelling*.

3.1. Theories for Event-B Mathematical Extensions for Hybrid Systems Modelling

The nature of hybrid systems requires to handle continuous and discrete features at the same level during modelling and proof. The Event-B extension mechanism enables the definition of such features in a generic and reusable way. In particular, we have defined Event-B theories for handling reals, continuous functions, and differential equations as well as their associated properties. These theories have already been used in the development of several case studies [12, 14, 16]. The complete theories are available at `https://irit.fr/~Guillaume.Dupont/models.php`.

In the following, we present some of these theories, as our developments heavily rely on them.

Figure 2 present the general architecture for the defined theories, and how they are related to each other.

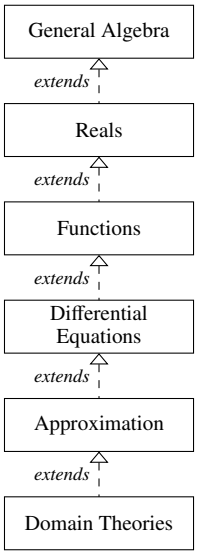

Figure 2: Theories Overall Architecture

Theories are categorised as follow:

- *General Algebra*: a set of theories that define the basic structures of general algebra: monoids, groups, rings, etc.;
- *Reals*: two theories that define the real numbers set with its associated operators and properties, as well as intervals;
- *Functions*: theories that bring additional structure and properties to functions, and in particular continuity and derivability, as well as the overall concept of piecewise defined functions;
- *Differential Equations*: a theory that defines the concept of differential equations as well as various useful related objects and properties to handle them;
- *Approximation*: a pair of theories that define the fundamental concepts needed to handle approximations, useful in the case of linearisation for example;
- *Domain Theories*: various theories for handling specific types of systems: cars, water tanks, robots, etc.

All of the above theories, except approximation, have been set up in the study developed in this paper. Regarding domain theories, we have defined the relevant theories related to the description of the shape of specific liquid tanks.

The next section shows an extract of the theories allowing to model differential equations. Only the relevant concepts necessary to understand this paper are introduced.

3.1.1. Overview of Relevant Hybrid Modelling Features.

THEORY
TYPE PARAMETERS E, F
DATA TYPES
DE(F)
CONSTRUCTORS
ode$(\mathrm{fun} : \mathbb{P}(\mathbb{R} \times F \times F), \mathrm{initial} : F, \mathrm{initialArg} : \mathbb{R})$
OPERATORS
solutionOf ***predicate*** $(D_R : \mathbb{P}(\mathbb{R}), \eta : \mathbb{R}^+ \pfun F, \mathrm{eq} : \mathbf{DE}(F))$
Solvable ***predicate*** $(D_R : \mathbb{P}(\mathbb{R}), \mathrm{eq} : \mathbf{DE}(F))$
direct definition
$\exists x \cdot x \in (\mathbb{R}^+ \pfun F) \land \mathbf{solutionOf}(D_R, x, \mathrm{eq})$
...
AXIOMS
CauchyLipschitz :
$\forall \mathcal{E}, D, D_F \cdot \mathcal{E} \in \mathbf{DE}(F) \land \ldots \Rightarrow \mathbf{Solvable}(D, \mathcal{E})$
...
END

Listing 2: Differential Equation Theory Snippet

Modelling hybrid systems requires the introduction of multiple specific features which are defined below.

- **DE**(S) type for differential equations for which solutions are valued in set S;
- **ode**(f, η_0, t_0) represents the *Ordinary Differential Equation* (ODE) $\dot{\eta}(t) = f(\eta(t), t)$ with initial condition $\eta(t_0) = \eta_0$ (note that we could define other constructors for other types of dynamics);
- **solutionOf**($D, \eta, \mathcal{E}$) is the predicate stating that function η is a solution of equation $\mathcal{E}$ on subset D;
- **Solvable**($D, \mathcal{E}, H$) is the predicate stating that equation $\mathcal{E}$ has a solution defined on subset D so that the solution satisfies the constraint H;
- An encoding of the *Cauchy-Lipschitz theorem*, that allows to demonstrate the solvability of a given equation under some specific conditions;

These features have been encoded in a theory from which we show a snippet in Listing 2 (the theory accumulates more than 150 operators and 350 properties).

Other, more specialised expressions and predicates are defined (*FlowEquation*, *FlowODE*) in additional theories. Note that all these definitions use *algebraic datatypes* together with axioms, theorems and proof rules.

3.1.2. System State Modelling

Modelling hybrid systems requires the definition of state variables to characterise both discrete and continuous concepts associated with a system. The definition of the continuous state variables is time dependent.

Time. A notion of time is needed to define continuous behaviour. We thus introduce dense time $t \in \mathbb{R}^+$, modelled as a continuously evolving variable.

System state. The state of the system is represented by variables. Because of its hybrid nature, we need two types of variables in order to represent its discrete and continuous aspects:

- **discrete variables** $x_s \in STATES$, that represent the controller's state (or *mode automaton*). They evolve in a point-wise manner, through instantaneous changes.
- **continuous variables** $x_p \in \mathbb{R}^+ \to S$, that represent the plant's physical quantities, and evolve continuously. They are modelled as functions of time, valued in S.

3.1.3. State Variables Assignment

Continuous variables are essentially functions of time and are at least defined on $[0, t[$ (where t is the current time). Updating such variables thus requires to 1) make the time progress from t to $t' > t$, and 2) append to the already existing function a new piece corresponding to its extended behaviour (on $[t, t']$) while ensuring its "past" (i.e. whatever happened during $[0, t[$) remains unchanged.

Similarly to the classic Event-B's before-after predicate (*BAP*), we define a *continuous before-after predicate* (*CBAP*) operator, denoted $:|_{t \to t'}$, as follows:

$$x_p :|_{t \to t'} \mathcal{P}(x_s, x_p, x'_p) \;\&\; H \equiv \; [0, t[\lhd x'_p = [0, t[\lhd x_p \qquad (PP)$$

$$\wedge\; \mathcal{P}(x_s, [t, t'] \lhd x_p, [t, t'] \lhd x'_p) \qquad (PR)$$

$$\wedge\; \forall t^* \in [t, t'], x'_p(t^*) \in H \qquad (LI)$$

We use the notation $CBAP(x_s, x_p, x'_p)$ as an abbreviation for $PP(x_p, x'_p) \wedge PR(x_s, x_p, x'_p) \wedge LI(x_p, x'_p)$. The operator consists of 3 parts: past preservation and coherence at assignment point (*PP*), before-after predicate on the added section (*PR*), and local invariant preservation (*LI*). The discrete state variables x_s do not change in the interval $[t, t']$ but the predicate $\mathcal{P}$ may use them for control purposes.

Note that this operator is well-defined if and only if $t' > t$, as otherwise the interval $[t, t']$ would not be well-defined.

From the above definition, shortcuts can be introduced for readability purposes:

- Continuous assignment: $x :=_{t\to t'} f \;\&\; H \;\equiv\; x :|_{t\to t'} x' = f \;\&\; H$
- Continuous evolution along a solvable differential equation $\mathcal{E} \in \mathbf{DE}(S)$: $x :\sim_{t\to t'} \mathcal{E} \;\&\; H$ $\equiv\; x :|_{t\to t'} \mathbf{solutionOf}([t, t'], x', \mathcal{E}) \;\&\; H$

Note that, similarly to Event-B's BAP assignment operator, the CBAP operator is designed to be fundamentally non-deterministic and with as few constraints as possible, so that it can be used efficiently and conveniently in conjunction with refinement.

Intuitively, non-determinism in the system's behaviour enables to delay some design decisions: we can describe the dynamics of an abstract model using high-level constraints (e.g. *"dynamics shall be expressed as the decreasing solutions of a solvable differential equation"*), and then provide actual concrete dynamics in a later refinement, making sure they verify these constraints (e.g. *"dynamics are given by* $\dot{x}_p = -1$*"*).

In practice, this permits the use of event parameters in models (that can be later refined with concrete values using witnesses), and also allows us to use powerful existence theorems, that may not be conclusive with regard to solution uniqueness (e.g. Peano-Lindelöf theorem).

In addition, and contrary to other approaches (e.g. Hybrid Event-B and Hybrid Programs), no assumptions are made about the nature of the dynamics used in the CBAP. This enables the user to propose any kind of behaviours, including rich constructs or even unconventional edge-cases, without having to modify the method in any way, and only by defining/integrating the appropriate theories. However, proof must still be performed, which may be difficult in the case of particularly complex dynamics and depending on the theories defining these dynamics.

3.2. A Generic Event-B Model for Hybrid Systems

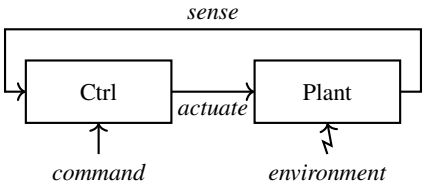

Figure 4: Hybrid System Global Architecture

Many hybrid systems behaviour and structure follow the pattern of Figure 4, called *controller-plant loop* configuration. In this setting, a hybrid system consists of a discrete controller (*Ctrl*) controlling a plant (*Plant*) characterised by continuous behaviour. The controller senses data issued from the plant, and may influence the plant behaviour through of actuators (e.g. motors, pumps, etc.). The controller may also retrieve commands from a user or another controller, and the plant itself is subject to environmental conditions, for instance wind, temperature, etc.

Following this pattern, inline with the definitions of Sections 3.1.1 and 3.1.2, we propose a generic Event-B model of hybrid systems. This model is at the core of our work: other development patterns (such as the architectural patterns presented in this paper) are formalised as refinements of this generic model.

VARIABLES t, x_s, x_p
INVARIANTS
inv1: $t \in \mathbb{R}^+$
inv2: $x_s \in$ STATES
inv3: $x_p \in \mathbb{R}^+ \nrightarrow S$
inv4: $[0, t] \subseteq \mathrm{dom}(x_p)$

Listing 3: Generic Time, Discrete and Continuous States

Model state. The generic model deals with three variables. x_s represents the controller's discrete state (associated to *Ctrl* in the diagram) that belongs to the *STATES* set consisting of the states of the system's mode automaton.

x_p is the system's continuous state (corresponding to the *Plant*'s state). It is a function of time (`inv3`) valued in the (continuous) *state space* S, usually $\mathbb{R}^n$. It represents the physical quantities that are sensed and/or controlled. Last, we recall that variable t models the physical, dense time.

Model behaviour. The defined model follows the control-command principle depicted in Figure 4. Two categories of events are defined. Discrete events are instantaneous, in the sense that time does not progress while they are being triggered. They are associated with changes in the state of the mode automaton either internal (*Transition* event) or induced by the sensing of the plant's state (*Sense* event). Continuous events, on the contrary, are associated to the passing of time. They describe the Plant's behaviour, either following environmental changes (*behave* event) or caused by actuation (*actuate* event). Note that all these generic events will be refined later for developing particular hybrid systems.

3.3. Discrete Events

Transition
ANY s
WHERE
grd1 : $s \in \mathbb{P}1(\text{STATES})$
THEN
act1 : $x_s :\in s$
END

Sense
ANY s , p
WHERE
grd1 : $s \in \mathbb{P}1(\text{STATES})$
grd2 : $p \in \mathbb{P}(\text{STATES} \times \mathbb{R} \times S)$
grd3 : $(x_s \mapsto t \mapsto x_p(t)) \in p$
THEN
act1 : $x_s :\in s$
END

Listing 4: Generic Transition and Sense Events

Transition. Transition events (corresponding to *command* arrow and the *Ctrl* box of Figure 4) model internal changes in the controller. They represent user commands, internal timers or non-deterministic choices that occur in the discrete part of the system (mode automata). It updates the state of the automaton (`act1` of `Transition` on Listing 4).

Sense. Sensing events (corresponding to *sense* arrow of Figure 4) model changes in the controller induced by the reading of the plant's state, generally obtained from sensors. As they are fired according to the plant's state and to the mode automaton's state, they are guarded by a predicate over $x_p(t)$ and x_s (`grd3` of `Sense` on Listing 4). The purpose is to change state x_s of the mode automaton (action `act1`).

3.4. Continuous Events

Behave
ANY eq , t'
WHERE
grd1 : $eq \in DE(S)$
grd2 : **Solvable**$([t, t'], eq, \top)$
THEN
act1 : $t, x_p :\sim_{t \to t'} eq \,\&\, \top$
END

Actuate
ANY eq , s , H , t'
WHERE
grd1 : $eq \in DE(S)$
grd2 : **Solvable**$([t, t'], eq, H)$
grd3 : $s \subseteq$ STATES
grd4 : $x_s \in s$
grd5 : $H \subseteq S$
grd6 : $x_p(t) \in H$
THEN
act1 : $t, x_p :\sim_{t \to t'} eq \,\&\, H$
END

Listing 5: Generic Behave and Actuation Events

Behave. Behave events (corresponding to the *environment* arrow of Figure 4) represent changes in the plant due to the environment: rain, wind, etc. They are encoded as shown on the left-hand side of Listing 5.

These events enforce, in action `act1`, the dynamics of the plant to comply with a differential equation under solvability condition (`gdr2`) but without any condition on the state of the mode automaton.

Actuate. Actuation events (corresponding to the *actuate* arrow of Figure 4) model changes in the plant induced by the controller (generally performed by actuators). They are encoded as on the right-hand side of Listing 5.

These events enforce, in action `act1`, the dynamics of the plant to comply with a differential equation under solvability condition (`gdr2`) and a constraint H on the plant evolution domain (`gdr5` and `gdr6`). Moreover, unlike for *Behave*, since *Actuate* results from a change in the controller, it is guarded by a predicate on the mode automaton (`gdr4`).

As mentioned above, both *Behave* and *Actuate* are continuous events. They rely on the continuous evolution operators defined in Section 3.1.3. Both events enforce plant behaviour by setting up a corresponding differential equation. Because these events are based on the CBAP operator $:|_{t\to t'}$ (and its derivatives), they are associated to a *feasibility* proof obligation, that ensures there exists (t', x'_p) such that $x_p :|_{t\to t'} \mathcal{P}(x_s, x_p, x'_p)$ & H holds. Establishing feasibility of a particular predicate in a refinement is completely tied to the nature of the predicate. When using differential equations, solvability needs to be asserted (by virtue of guard `gdr2`) using specific theorems such as the Cauchy-Lipschitz theorem, or any other property associated to the considered differential equation.

3.5. Semantics

The semantics of hybrid models we use is close to the one of Hybrid Event-B [5], hybrid programs in [2] or continuous action systems [8, 37].

In classical Event-B semantics, each model is associated with a discrete state-transition system, in which transitions are the fired machine events and states consist of the machine's variables. A system is hence characterised by a set of admissible traces i.e. a set of fired events that abide by the system's invariants.

In our approach, discrete events are timeless, while continuous ones are associated to the passing of time. In order to properly handle the modelling of continuous behaviour, the semantics of Event-B is enhanced to handle modelling of continuous phenomena which are, in nature, different from discrete behaviour. We have identified two categories of events: discrete (instantaneous) events, which use discrete assignment operators such as :| and := and continuous (not instantaneous) events that span over some duration and use continuous assignment operators, namely $:|_{t\to t'}$ and $:=_{t\to t'}$. Note that, if several (continuous or discrete) event guards are enabled, these enabled events are fired non-deterministically.

A model is then defined as follows. After initialisation, continuous events (*Behave* and *Actuate* events) execute continuously unless a discrete, instantaneous event is enabled (either a *Sense* or a *Transition* event). In this case, discrete events are preemptive, meaning they are executed before the control is handed over to continuous events. This protocol ensures that when the conditions (events' guards) are met, the controller is able to trigger control actions (*Sense* or *Transition*) that may or may not change the continuous behaviour of the plant (through triggering an *Actuate* event). Unlike *Actuate*, the *Behave* event neither requires a control action to be triggered nor any plant evolution constraint H. Sensing actions using the *Sense* event will re-establish the correct plant behaviour via the control loop in order to further trigger an *Actuate* event.

Note that the proposed semantics is the one proposed by R. Banach for Hybrid Event-B [5, 21], where discrete events correspond to *mode* events, while continuous events correspond to *pliant* events.

In our approach, we have formalised the features of Hybrid Event-B within Event-B, in order to offer an operational framework relying on the Rodin platform – which is missing for Hybrid Event-B. Proceeding in this way, we can use Rodin to model hybrid systems with both discrete and continuous behaviours, including support for model development and proofs.

Time-triggered vs. event-triggered. The semantics we proposed is so-called *event-triggered*: discrete phenomena are considered timeless events, that may occur at any time and interrupt continuous phenomena. This kind of semantics is relatively abstract and simplifies a number of problems, and in particular *zero-crossing*.

Formally, zero-crossing is when a controller "misses" events, i.e. when the continuous variable *crosses* a domain (guard, evolution domain) without the controller detecting it. With event-triggered semantics, we can always assume that the controller notices the continuous variable crossing the domain *at the exact time point it does so*.

Comparatively, time-triggered semantics is less abstract: it requires that the controller is always able to detect variable crossing at the very latest when it occurs (and ideally before). Time-triggered semantics is harder to prove but easier to implement. In practice, one may derive a concrete time-triggered model from an abstract event-triggered

one (with a refinement step, for instance), as to benefit from the easier proof with event-triggered semantics while approaching a level of abstraction closer to implementation.

Zeno behaviour. The duration of continuous events may be arbitrarily small, and even infinitely small, so that two discrete events are infinitely close. This may lead to a situation known as Zeno paradox, where an infinite number of discrete events is triggered in a finite amount of time.

3.6. The Generic Model in Rodin

The generic model is the entry point for the method. Specific hybrid system models are obtained by refining it, providing the various witnesses issued from event parameters and substituted variables. In itself, this model generates 13 proof obligations that are easily discharged. Among them there is an important obligation stating that if equation e is solvable then $x :\sim_{t\to t'} e$ is feasible.

This approach has been successfully applied to various case studies. [12, 14] show a class of systems with one controller and one plant while [16] demonstrates the possible use of the method for a system with one controller and several plants.

Models for the generic approach, including the above-mentioned case studies can be found at `https://www.irit.fr/~Guillaume.Dupont/models.php`.

4. A Case Study

To demonstrate how the various architectural patterns we propose are applied to model specific hybrid systems, we have chosen a case study that is classically exemplified in control theory literature. The objective of the case study is to control the volume of liquid in one or more tanks. In particular, the defined control must keep the volume level between given minimum and maximum levels. We identified the following cases based on the defined patterns.

- *Mono control* system where one tank is controlled by a unique controller;
- *Centralised control* where multiple tanks are controlled by a single controller;
- *Distributed control* where a set of communicating controllers, each of which controls one tank, that contribute to the control of the cumulative volumes of each tank.

In the next two sections, we first describe the case study and highlight two safety requirements. Then, a domain theory axiomatising the general characteristics and properties of tanks is presented. This theory provides relevant properties used in further developments.

4.1. Description of the Case Study

Figure 7 depicts an abstract representation of the case study we use throughout this paper.

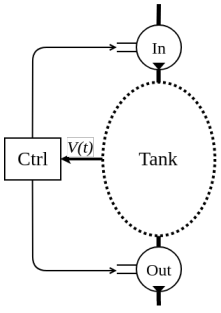

Figure 7: Abstract Tank Model

The system consists of tank(s) containing a volume of liquid, of arbitrary shape and architecture (abstracted). It is connected to a generic controller, that is able to actuate pumps (an input and an output pump) to fill or empty the tank(s). The connected controller can also sense the volume in the tank(s).

The goal of the system is to maintain the volume of liquid inside the tank(s) between too given constants V_{low} and V_{high} (**SAF1**), and also to ensure that the flow in the tank is bounded by a constant ΔV_{max}, in order to avoid turmoil (**SAF2**). Formally

$$\forall t, V_{low} \leq V(t) \leq V_{high} \wedge |\dot{V}(t)| \leq \Delta V_{max}$$

Last, we suppose that activating the input pump will fill up the tank(s), and activating the output pump will empty it. Also, when no pump is active, the volume in the tank does not vary. Apart from that, there is no additional hypothesis on the system at this level of description.

4.2. A Theory of Water Tanks

Water tanks are associated with a number of specific phenomena and properties: pumps physical modelling, specific differential equations, etc. These features have been formalised in a *domain specific theory* of water tanks.

The theory is split in two parts: a theory for *valves* and a theory for *flows*. Extracts for both these theories are given in Listing 6. They define a number of operators used in the model when instantiating the patterns, as well as several theorems and axioms, used during the proving process.

```
THEORY Valves IMPORT RReal
DATATYPES
  Status ≙ ValveOpen | ValveClosed
  InOutValve ≙ InOut( in_status : Status ,
                      out_status : Status)
OPERATORS
  closeIn expression (iov: InOutValve)
    recursive definition
      cases iov
        – InOut(i, o) ⇒ InOut(ValveOpen, o)
  ...
  in_rstatus expression (iov: InOutValve)
    recursive definition
      cases iov
        – InOut(ValveOpen, o) ⇒ 1.0
        – InOut(ValveClosed, o) ⇒ 0.0

  InOutPossible expression ()
    direct definition
        {InOut(ValveOpen, ValveOpen), ...}
  ...
```

```
THEORY Flow IMPORT Valves , DiffEq
DATATYPES
  TankState ≙ Stable | Emptying | Filling | Normal
OPERATORS
  isFlow predicate (s: TankState , D: ℙ(ℝ) ,
      Φ: ℝ ⇸ ℝ , Q_min: ℝ , Q_max: ℝ)
    well–definedness D ⊆ dom(Φ)
    recursive definition
      cases s
        – Stable ⇒ constant(D, Φ)
        – Emptying ⇒ decreasing(D, Φ) ∧ Q_min ≤ Φ ≤ Q_max
  ...
  isFlowEq predicate (s: TankState , D: ℙ(ℝ) ,
      e: DE(S) , Q_min: ℝ , Q_max: ℝ)
    well–definedness Solvable(D, e)
    direct definition
        ∀η · η ∈ ℝ ⇸ ℝ ∧ D ⊆ dom(η) ∧ solutionOf(D, η, e)
                    ⇒ isFlow(s, D, η, Q_min, Q_max)
  ...
  FlowODE expression (Q_min: ℝ , Q_max: ℝ ,
      δ^in: ℝ , δ^out: ℝ , io: InOutValve)
    well–definedness 0 < Q_min , 0 < Q_max , δ^in > 0 , δ^out > 0
    direct definition ...

  TankModeChange predicate (state: TankState , io_1 :
      InOutValve , io_2 : InOutValve)
    direct definition ...
```

Listing 6: Valves and Flow Domain Theories

Valves. The theory first defines the **Status** datatype as an enumeration specifying whether a valve is open (*ValveOpen*) or closed (*ValveClosed*). Secondly, the theory defines the **InOutValve** product type, which includes the status of two valves (an input and an output one). This type is associated with a single constructor, *InOut*, which allows both statuses to be set.

These types are associated with convenient valve opening and closing operators (*closeIn*, *openOut*, etc.). The theory also provides a way to associate a real number with the status of the valve, *ValveClosed* being associated with 0 and *ValveOpen* with 1. Finally, the theory defines the set of possible valve combination, *InOutPossible*.

Flow. This second theory is based on the previous one and on the differential equation theory. It defines several operators to characterise the various features of any tank. In particular, it proposes an enumeration datatype for the possible tank states **TankState** (see Section 5.2.1). It also makes accessible various operators to handle the generic concept of *flow*.

The predicate *isFlow* is defined, that is true for a domain $D \subseteq \mathbb{R}$, a state $s \in$ **TankState**, a function $\Phi \in \mathbb{R} \nrightarrow \mathbb{R}$ with $D \subseteq \text{dom}(\Phi)$ and two bounds Q_{min} and Q_{max} (in $\mathbb{R}$) if and only if Φ *behaves according* to s on D, and is bounded by Q_{min} and Q_{max}.

In this context, *behaving according to s* typically means that Φ is decreasing on D if state s is *Emptying*, constant if state is *Stable* and so on.

This predicate is adapted to be used on differential equation (*isFlowEq*), to check that the solutions of given equation are adequate with regard to the given state. This operator is further specialised to ODE, since the ODE's function's sign

gives the monotonicity of the solution. Additionally, the theory gives a simple ODE for modelling the flow of a tank (*FlowODE*, corresponding to the one presented in Section 6.3). It also proposes an ODE for a tank with no flow, used for initialisation. In addition, the *TankModeChange* predicate is defined for the particular case of two tanks. It links a global state (*state*) representing a global view of the two tanks to the local states of each tank (io_1 and io_2). It is studied more thoroughly in Section 6.3.1.

Finally, we define various theorems aimed at handling the defined operators, especially the ODEs and their solvability.

The following sections address this case study, under different hypotheses, and most importantly using different architectures, to illustrate the formalised architectural patterns we propose in this paper.

5. Single-to-Single Architectural Pattern

The single to single architectural pattern is the basic one. It represents the case of one controller for one plant. This pattern represents the basic pattern from which all the others are derived by refinement.

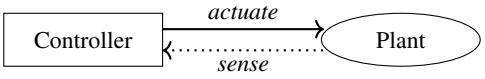

Figure 9: Single-to-single Pattern Typical Scenario

5.1. The S2S pattern

The *single-to-single* (S2S for short) architectural pattern is depicted in Figure 9. It is the general structure of hybrid systems handled by the generic model, and consists of a single controller, controlling a single plant.

This pattern corresponds to refining and instantiating the generic model for a given hybrid system. It is not described in this section, as it has been detailed in Section 3. Next section shows how it is instantiated to model the case study of a single controller for a single tank.

5.2. Application to Case Study – Abstract Tank Model

The first step for addressing this case study is to model the abstraction of the system depicted in Figure 7. Formally, the abstract tank model is a refinement of the generic model.

5.2.1. Preliminary Study

Plant description. The controlled plant is an abstract tank containing a volume $V(t) \geq 0$ of liquid. This tank is finite and can contain a maximum volume of $V_{max} > 0$. The plant is tied to two pumps, one input pump able to fill-up the tank (i.e. increase $V(t)$) and another to empty it (i.e. decrease $V(t)$). Note that the behaviour of $V(t)$ is unspecified when both pumps are operating at the same time.

At this point, no hypothesis is made on the shape of the tank, and no hypothesis is made on the exact behaviour of the pumps, except that 1) when the input pump is open the volume increases, 2) when the output pump is open the volume decreases, and finally 3) when no pump is open the volume does not change.

Consequently, for this model, the differential equation describing the volume's behaviour is *unknown*, although it is constrained by the hypotheses on the pumps' behaviour.

Note that, for convenience, it is assumed that tank initially contains some arbitrary initial volume V_0, such that $V_{low} \leq V_0 \leq V_{high}$.

Controller description. The controller's task is to actuate the input and output pumps, in order to fill or empty the tank. Concretely, it operates in 4 modes, and switches from one to another when needed, to preserve safety. These modes are:

- *stable*, the volume does not change (i.e. V is a constant function). Provided V remains within the bounds, the controller can always switch to this mode;

- *emptying*, the volume is decreasing. The controller may switch to this mode if $V > V_{low}$, and it switches automatically to it when $V = V_{high}$;
- *filling*, the volume is increasing. The controller may switch to this mode if $V < V_{high}$, and it switches automatically to it when $V = V_{low}$;
- *normal*, the volume varies arbitrarily. The controller may switch to this mode if V is within the bounds. It is a default mode, where liquid is flowing freely.

These modes induce 4 transition events, allowing to visit each state, and 2 sensing events, that detect when the volume reaches a bound and puts the system in the corresponding correcting mode. Note that, when actuating the pump, the controller must make sure that the volume level does not change too rapidly, i.e. its derivative is bounded ($\dot{V} \leq \Delta V_{max}$).

Requirements. In summary, the system requirements are described as follows:

FUN1 the volume varies in the tank, it is decreasing in *emptying* mode, increasing in *filling* mode, constant in *stable* mode, and safely varying in *normal* mode;

ENV1 the volume is physically bounded by 0 and some constant V_{max}: $\forall t \in \mathbb{R}^+, 0 \leq V(t) \land V(t) \leq V_{max}$;

SAF1 the volume must always remain within the set bounds, V_{low} and V_{high}: $\forall t \in \mathbb{R}^+, V_{low} \leq V(t) \land V(t) \leq V_{high}$;

SAF2 the variation of the volume ($\dot{V}(t)$) is always below the maximum allowed variation ΔV_{max}: $\forall t \in \mathbb{R}^+, |\dot{V}(t)| \leq \Delta V_{max}$.

5.2.2. Event-B Development

The preliminary study for this system leads to the definition of an Event-B model. This first model (the *abstract tank*) is based on the generic model presented in Section 3.

CONTEXT AbstractTankCtx **EXTENDS** GenericCtx
CONSTANTS V_{max}, V_{low}, V_{high}, ΔV_{max}, V_0
AXIOMS
axm1: $S = \mathbb{R}$
axm2: partition($STATES$, {$Stable$}, {$Emptying$}, {$Filling$}, {$Normal$})
axm3−6: $V_{max}, V_{low}, V_{high}, \Delta V_{max} \in \mathbb{R}$
axm7: $0 \leq V_{low} \leq V_{high} \leq V_{max}$
axm8: $\Delta V_{max} > 0$
axm9−11: $V_0 \in \mathbb{R} \land V_{low} \leq V_0 \land V_0 \leq V_{high}$
END

Listing 7: Abstract Tank – Context

Constants and axioms. The system requires a number of properties to be written (in addition to the theories of valves and flows presented in Section 4.2), that are identified in an Event-B context, given in Listing 7. This context defines the required constants with associated properties (`axm3-11`). We also define the system's state-space ($\mathbb{R}$ in `axm1`) as well as the controller modes (`axm2`), based on those defined in the tank theories (*flow* and *valves*).

Machine header. Listing 8 presents the machine's header and initialisation event. The header defines the variables of the system (mainly the continuous state V) with associated properties (`inv1-2`). `inv3` is the gluing invariant of the system (a substitution) while `inv4` encodes the environment property **ENV1** of the requirements section. Finally, `inv5` and `inv6` model the safety requirement **SAF1** and **SAF2** respecitvely.

The system's initialisation is straightforward. The controller starts in *stable* mode, and the volume is set to the constant V_0, initial volume contained in the tank.

MACHINE AbstractTank **REFINES** Generic
SEES AbstractTankCtx
VARIABLES t, x_s, V
INVARIANTS
inv1 : $V \in \mathbb{R} \nrightarrow S$
inv2 : $[0, t] \subseteq \text{dom}(V)$
inv3 : $V = x_p$
inv4 : $\forall t^* \cdot t^* \in [0, t] \Rightarrow 0 \leq V(t) \wedge V(t) \leq V_{max}$
inv5 : $\forall t^* \cdot t^* \in [0, t] \Rightarrow V_{low} \leq V(t^*) \wedge V(t^*) \leq V_{high}$
inv6 : $\forall t^* \cdot t^* \in [0, t] \Rightarrow |\dot{V}(t^*)| \leq \Delta V_{max}$

INITIALISATION
WITH
x'_p : $V' = x'_p$
THEN
act1 : $t := 0$
act2 : $x_s := Stable$
act3 : $V := \{0 \mapsto V_0\}$
END

Listing 8: Abstract Tank – Machine Header

ctrl_transition_normal REFINES Transition
WHERE
grd1 : $V(t) < V_{high}$
grd2 : $V_{low} < V(t)$
WITH
s : $s = \{Normal\}$
THEN
act1 : $x_s := Normal$
END

ctrl_sense_too_high REFINES Sense
WHERE
grd1 : $V_{high} \leq V(t)$
WITH
s : $s = \{Emptying\}$
p : $p = STATES \times \mathbb{R} \times \{V^* \mid V_{hight} \leq V^*\}$
THEN
act1 : $x_s := Emptying$
END

Listing 9: Abstract Tank – Transition and Sensing

Discrete events. Listing 9 gives a transition and sensing events from the system. `ctrl_transition_normal` is a transition event that allows the controller to move to *normal* mode. It is guarded by a condition to preserve safety in this mode.

The sensing event `ctrl_sensing_too_high` detects when the volume reaches V_{high} (see `grd1`), and causes the controller to move to *emptying* mode.

For both events, witnesses (for instantiation) are provided for s and p, reflecting their guards and target state.

Continuous events. Listing 10 presents the actuation of the system. It refines the abstract `Actuate` event and constrains it in order to ensure the system's invariant holds.

EVENT ctrl_actuate_pumps REFINES Actuate
ANY eq, s, t'
WHERE
grd0 : $t' \in \mathbb{R} \wedge t < t'$
grd1 : $eq \in \mathbf{DE}(S)$
grd2 : $\mathbf{Solvable}([t, t'], eq, \{V* \mid V_{low} \leq V^* \wedge V^* \leq V_{high}\}$
grd3 : $isFlowEq(s, [t, t'], eq, 0, V_{max})$
grd4 : $s \in STATES$
grd5 : $x_s = s$
grd6 : $V_{low} \leq V(t) \wedge V(t) \leq V_{high}$
WITH
x'_p : $x'_p = V'$
H : $H = \{V^* \mid V_{low} \leq V^* \wedge V^* \leq V_{high}\}$
THEN
act1 : $V :\sim_{t \to t'} eq \,\&\, \{V* \mid V_{low} \leq V^* \wedge V^* \leq V_{high}\}$
END

Listing 10: Abstract Tank – Actuation Event

For instantiation purposes, in the `WITH` clause, x_p is substituted with V using the gluing invariant, and a concrete evolution domain H is given to enforce the volume V to remain within the given bounds V_{low} and V_{high}. At this level,

the differential equation is still unknown, but it is constrained by the required behaviour of the pumps. These constraints are encapsulated in the *isFlowEq* predicate (see Section 4.2). Therefore, according to the shape of the concrete tank, any differential equation, modelling the liquid behaviour of this concrete tank, that fulfils the conditions expressed by guards `grd2` and `grd3` may be used as a witness for the parameter `eq` at instantiation.

5.2.3. Proofs

At this point, proofs are straightforward. The model generated 64 proof obligations, most of them (around 30%) coming from well-definedness. Simulation and guard strengthening POs (44%) are discharged by exploiting the fact that the events are built by substituting variables/parameters.

Last, invariant proofs (23%), putting apart trivial type-related invariants, rely on the specific constraints given to the model (values for H and use of the *isFlowEq* predicate), together with the properties these constraints entail, that are given in the tank theories. The remaining POs are related to feasibility (3%).

The *single-to-single* model serves as a base for the application of the *single-to-many* and *many-to-many* patterns, as both plant (*single-to-many*) and controller (*many-to-many*) are refined.

6. Single-to-Many Architectural Pattern

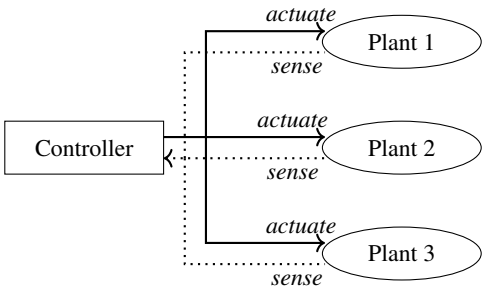

Figure 14: Single-to-many Pattern Typical Scenario

The *single-to-many* (S2M for short) pattern appears in scenarios such as the one shown in Figure 14. It represents *centralised control* systems: multiple plants are controlled by a single controller. It is able to access each plant's state, and influence each plant using separate actuators. In this setup, the difficulty is to enforce a global property on the system, while having separate plants, each with its own properties.

As a simple example of this kind of situation, one can imagine a set of tanks in which the valves are controlled by a single system. The goal is to maintain, as invariant, a certain volume of liquid between low and high volume levels as a whole (i.e. globally for all the tanks) by opening the output and the input valves as needed. The difficulty of controlling this system comes from the ability to change the volume of multiple tanks while enforcing this invariant (for example, we could have several completely empty tanks while still retaining the property).

6.1. Pattern Model

This pattern is presented as a refinement, showing an abstract generic model (denoted as a superscript A) and a refinement (denoted as a superscript C) side by side.

As a matter of simplification, it is assumed that the concrete machine handles 2 continuous state variables (i.e. two plants), but it can easily be extended to use as many continuous state variables as necessary. We also consider that the discrete state does not change from one machine to the other.

6.1.1. Machine Headers

Listing 11 gives the header of the two machines. Following the generic model presented in Section 3, the machines first define a variable for time (t) and a variable for the discrete state (x_s) with associated invariants. Machine M^A

MACHINE M^A
VARIABLES t, x_s, x_p^A
INVARIANTS
inv1 : $t \in \mathbb{R}^+$
inv2 : $x_s \in \text{STATES}$
inv3 : $x_p^A \in \mathbb{R} \nrightarrow S^A$
inv4 : $[0, t] \subseteq \text{dom}(x_p^A)$

MACHINE M^C **REFINES** M^A
VARIABLES t, x_s, $x_{p,1}^C$, $x_{p,2}^C$
INVARIANTS
inv31 : $x_{p,1}^C \in \mathbb{R} \nrightarrow S_1^C$
inv32 : $x_{p,2}^C \in \mathbb{R} \nrightarrow S_2^C$
inv41 : $[0, t] \subseteq \text{dom}(x_{p,1}^C)$
inv42 : $[0, t] \subseteq \text{dom}(x_{p,2}^C)$
inv5 : $x_p^A = f(x_{p,1}^C, x_{p,2}^C)$

Listing 11: S2M Pattern – Machine Header

defines a continuous state x_p^A valued in S^A and machine M^C defines two continuous states $x_{p,1}^C$ and $x_{p,2}^C$ valued in S_1^C and S_2^C respectively.

The key point here is `inv5` of M^C, which is the *gluing invariant* of the refinement. It states that, on interval $[0, t]$, x_p^A, $x_{p,1}^C$ and $x_{p,2}^C$ are linked by function $f \in (S_1^C \times S_2^C) \rightarrow S^A$.

INITIALISATION^A
THEN
act1 : $t := 0$
act2 : $x_s :\in \text{STATES}$
act3 : $x_p^A :\in \{0\} \rightarrow S^A$
END

INITIALISATION^C **REFINES** **INITIALISATION**^{A}
WITH $x_p^{A\prime}$: $x_p^{A\prime} = f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime})$
THEN
act1 : $t := 0$
act2 : $x_s :\in \text{STATES}$
act31 : $x_{p,1}^C :\in \{0\} \rightarrow S_1^C$
act32 : $x_{p,2}^C :\in \{0\} \rightarrow S_2^C$
END

Listing 12: S2M Pattern – Initialisation

The initialisation events for both machines are shown in Listing 12. Here, x_p^A is substituted using the gluing invariant `inv5` and "disappears" thanks to refinement. It is substituted by the new continuous state $x_{p,1}^C, x_{p,2}^C$. In addition, it is subsequently substituted in the actions by these new variables, and based on the gluing invariant, a witness is provided for $x_p^{A\prime}$. This witness allows to maintain the invariant.

6.1.2. Discrete Events

It is assumed that the discrete behaviour of the system does not change from the abstract machine to the concrete one (i.e. the controller remains single). This means that *transition* events remain unchanged. *Sensing* events, on the other hand, need to be addressed as they have access to a continuous state.

Sense^A
ANY s, p^A
WHERE
grd1 : $s \in \mathbb{P}1(\text{STATES})$
grd2 : $p^A \in \mathbb{P}(\text{STATES} \times \mathbb{R} \times S^A)$
grd3 : $(x_s \mapsto t \mapsto x_p^A(t)) \in p^A$
THEN
act1 : $x_s :\in s$
END

Sense^C **REFINES** **Sense**^{A}
ANY s, p^A
WHERE
grd1 : $s \in \mathbb{P}1(\text{STATES})$
grd2 : $p^A \in \mathbb{P}(\text{STATES} \times \mathbb{R} \times S^A)$
grd3 : $(x_s \mapsto t \mapsto f(x_{p,1}^C(t), x_{p,2}^C(t))) \in p^A$
THEN
act1 : $x_s :\in s$
END

Listing 13: S2M Pattern – Sensing Event

Listing 13 presents the sensing events for the S2M pattern. Note that the event is unchanged, except for the abstract state x_p^A that is substituted with the concrete state $(x_{p,1}^C, x_{p,2}^C)$ by exploiting the particular shape of the gluing invariant.

This replacement is sound (see Section 6.2.2) and the resulting guard is *equivalent* to the guard of the abstract sensing event. Note that, as for any event refinement, it is possible to propose a stronger guard provided that this event remains feasible.

6.1.3. Continuous Events

As the refinement of the S2M pattern affects the system's continuous state, continuous events provide important updates. In this section, we only address the case of the *actuation* event, since *behave* can be seen as a simpler (i.e. less constrained) version of the actuation event.

Actuate^A
ANY $\mathcal{P}^A$, s , H^A , t'
WHERE
grd0 : $t' > t$
grd1 : $\mathcal{P}^A \in (\mathbb{R}^+ \nrightarrow S^A) \times (\mathbb{R}^+ \nrightarrow S^A)$
grd2 : **Feasible**$(x_p^A, [t, t'], \mathcal{P}^A, H^A)$
grd3 : $s \subseteq$ STATES
grd4 : $x_s \in s$
grd5 : $H^A \subseteq S^A$
grd6 : $x_p^A(t) \in H^A$
THEN
act1 : $x_p^A :|_{t \to t'} \mathcal{P}^A(x_p^A, x_p^{A\prime})$ & H^A
END

Actuate^C REFINES Actuate^A
ANY $\mathcal{P}^C$, s , H^A , t'
WHERE
grd0 : $t' > t$
grd1 : $\mathcal{P}^C \in (\mathbb{R}^+ \nrightarrow (S_1^C \times S_2^C)) \times (\mathbb{R}^+ \nrightarrow (S_1^C \times S_2^C))$
grd2 : **Feasible**$(x_{p,1}^C \otimes x_{p,2}^C, [t, t'], \mathcal{P}^C, f^{-1}[H^A])$
grd3 : $s \subseteq$ STATES
grd4 : $x_s \in s$
grd5 : $H^A \subseteq S^A$
grd6 : $f(x_{p,1}^C, x_{p,2}^C) \in H^A$
WITH
$x_p^{A\prime}$: $x_p^{A\prime} = f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime})$
THEN
act1 : $x_{p,1}^C, x_{p,2}^C :|_{t \to t'} \mathcal{P}^C(x_{p,1}^C \otimes x_{p,2}^C, x_{p,1}^{C\prime} \otimes x_{p,2}^{C\prime})$ & $f(x_{p,1}^C, x_{p,2}^C) \in H^A$
END

Listing 14: S2M Pattern – Actuation Event

Listing 14 shows the `Actuate` event of the S2M pattern. Notice the use of the direct product ($\otimes$) to bind together two functions, in order to be able to use the theory's operators on them.

The only parameter to be changed is $\mathcal{P}^A$, replaced by $\mathcal{P}^C$. In theory, we would have to provide a witness, but since this is a pattern, it is expected that, at this point, these predicates would actually contain definite values, and a witness would depend heavily on those values.

Local invariant H^A is unchanged; instead, guards `grd2` and `grd6` are updated to use the concrete continuous state $(x_{p,1}^C, x_{p,2}^C)$ instead of the abstract one, again exploiting the gluing invariant. The evolution domain of the CBAP operator (right-hand side of the &) is modified in a similar way. It is used for establishing simulation (see Section 6.2.3). Note that, equivalently, predicate of the form $f(x_{p,1}^C, x_{p,2}^C) \in H^A$ may be replaced by: $(x_{p,1}^C, x_{p,2}^C) \in f^{-1}[H^A]$.

Finally, a witness is provided for $x_p^{A\prime}$, recalling the gluing invariant.

6.2. Proofs

The *single-to-many* architectural pattern yields multiple proof obligations that need to be proved. Since it is a pattern, these proof obligations are general, and discharging them often requires relying on the particular shape and properties of the model parameters (predicates, variables, and so on).

In this section, we study the generated POs associated with the pattern, and give general techniques to discharge them.

6.2.1. Invariant Preservation

Gluing invariant preservation (PO 7 of Table 2b) shows the correctness of the relation between the abstract and the concrete variables of the system. In the case of the *single-to-many* refinement, the preservation of the *gluing invariant* (`inv5` of Listing 11) is ensured by the use of *witnesses* for $x_p^{A\prime}$, in particular in continuous events.

The preservation of the other invariants (PO 2 of Table 2a), specific to the machine, is to be proven on case-by-case basis. It highly depends on the nature of the invariant and of the system's behaviour.

6.2.2. Guard Strengthening

Guard strengthening (see PO 6 in Table 2b) is a proof obligation that is directly related to refinement, and thus is generated when instantiating the pattern. In our case, we address guard strengthening of guards that involve a continuous state, i.e. the guards of sensing events and the evolution domain consistency of actuation events.

For instance, guard strengthening for sensing events yields the following PO:

$$A \wedge I \wedge W \wedge x_p^A =_{[0,t]} f(x_{p,1}^C, x_{p,2}^C) \wedge (x_s \mapsto t \mapsto f(x_{p,1}^C(t), x_{p,2}^C(t))) \in p^A \Rightarrow (x_s \mapsto t \mapsto x_p^A(t)) \in p^A$$

Discharging this PO is straightforward, using the gluing invariant to substitute x_p^A with $f(x_{p,1}^C, x_{p,2}^C)$ on the right-hand side of the implication (by noticing that $t \in [0, t]$).

Similarly, we consider the PO associated to evolution domain consistency (`grd6` of actuation events):

$$A \wedge I \wedge W \wedge x_p^A =_{[0,t]} f(x_{p,1}^C, x_{p,2}^C) \wedge f(x_{p,1}^C(t), x_{p,2}^C(t)) \in H^A \Rightarrow x_p^A(t) \in H^A$$

Using the same technique as for guard strengthening POs in sensing events (i.e. substitution using the gluing invariant) allows to discharge the PO.

Note that the concrete and abstract guards are *equivalent*. It is always possible to propose a stronger guard in the refinement, provided it implies the abstract one (e.g. $f(x_{p,1}^C(t), x_{p,2}^C(t)) \in H \subseteq H^A$).

6.2.3. Simulation

Simulation (see PO 5 in Table 2b) is a crucial proof obligation associated with refinement. It ensures that the behaviour of a concrete event simulates the one of the abstract event.

This PO is written as:

$$\begin{aligned} A \wedge I \wedge G \wedge x_p^{A\prime} =_{[0,t']} f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime}) \wedge CBAP(t, t', x_{p,1}^C \otimes x_{p,2}^C, x_{p,1}^{C\prime} \otimes x_{p,2}^{C\prime}, \mathcal{P}^C, f^{-1}[H^A]) \\ \Rightarrow CBAP(t, t', x_p^A, x_p^{A\prime}, \mathcal{P}^A, H^A) \end{aligned}$$

We can unfold the definition of CBAP (as given in Section 3.1.3) on each side of the implication:

$$\begin{aligned} & A \wedge I \wedge G \wedge x_p^{A\prime} =_{[0,t']} f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime}) \\ & \quad \wedge [0,t[\lhd (x_{p,1}^{C\prime} \otimes x_{p,2}^{C\prime}) = [0,t[\lhd (x_{p,1}^C \otimes x_{p,2}^C) && \Rightarrow [0,t[\lhd x_p^{A\prime} = [0,t[\lhd x_p^A && (PP) \\ & \quad \wedge \mathcal{P}^C([0,t] \lhd (x_{p,1}^C \otimes x_{p,2}^C), [t,t'] \lhd (x_{p,1}^{C\prime} \otimes x_{p,2}^{C\prime})) && \quad \wedge \mathcal{P}^A([0,t] \lhd x_p^A, [t,t'] \lhd x_p^{A\prime}) && (PR) \\ & \quad \wedge (x_{p,1}^{C\prime} \otimes x_{p,2}^{C\prime}) \in_{[t,t']} f^{-1}[H^A] && \quad \wedge x_p^{A\prime} \in_{[t,t']} H^A && (LI) \end{aligned}$$

Using the gluing invariant and the witness ($x_p^{A\prime} =_{[0,t']} f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime})$) to substitute x_p^A and $x_p^{A\prime}$ in the formula, and by exploiting the properties of the domain restriction operator ($\lhd$), the proof for *PP* and *LI* are completed.

At this step, the *continuous predicate simulation* (CPSIM) part of the proof is left to prove:

$$\begin{aligned} A \wedge I \wedge G \wedge x_p^{A\prime} =_{[0,t']} f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime}) \wedge CBAP(t, t', x_{p,1}^C \otimes x_{p,2}^C, x_{p,1}^{C\prime} \otimes x_{p,2}^{C\prime}, \mathcal{P}^C, f^{-1}[H^A]) \\ \Rightarrow \mathcal{P}^A([0,t] \lhd x_p^A, [t,t'] \lhd x_p^{A\prime}) \end{aligned} \qquad (CPSIM)$$

Such proof is to be conducted on a case-by-case basis, as it highly depends on the form of the predicates $\mathcal{P}^A$ and $\mathcal{P}^C$. Evolution domain $f^{-1}[H^A]$ can be strengthened in order to give additional properties, used as hypotheses for this proof.

In the case where the event uses the $:\sim_{t \to t'}$ operator, the proof consists in establishing that the solutions of the concrete equations, once processed through the gluing relation f, yield a function that is a potential solution of the abstract differential equation.

6.3. Application to the Case Study

We propose a refinement of the abstract tank model developed in Section 5.2 into a system that consists of one controller controlling two separate tanks of definite shape (see Figure 19), using the *single-to-many* architectural pattern. The goal of the system remains the same, but the abstract tank is replaced by two *cylindrical tanks* of known bases and heights. The sensing is also updated to better reflect "real world" situations: only the height is accessible (e.g. thanks to a float), and the controller needs to derive the actual volume based on this height and the parameters of the tank.

Last, the pumps' behaviour is fixed, either fully open or fully closed, and delivers a fixed flow.

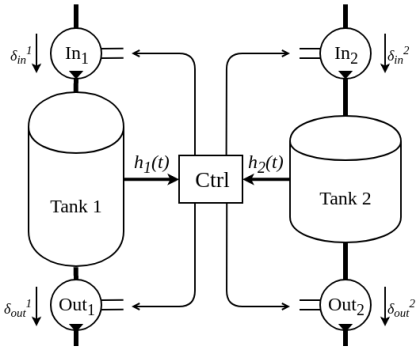

Figure 19: 1 Controller, 2 Tanks Configuration

6.3.1. Preliminary Study

Plant description. The two plants (Tank 1 and Tank 2) are controlled simultaneously. Each plant consists of a cylinder tank with associated pumps. The volume of each tank cannot be accessed directly; instead the controller senses the *height of liquid* inside, denoted $h_{1/2}(t)$.

Tank 1 has a base of B_1 and a maximum height of $H_{1,max} > 0$. It is connected to an input pump of flow $\delta_1^{in} > 0$ and an output pump of flow $\delta_1^{out} > 0$. Initially, it contains a height h_1^0 of liquid. Similarly, Tank 2 operates in a similar manner, replacing the index 1 by 2 in the given parameters.

Note that the volume $V_i(t)$ of tank i is given by the formula $V_i(t) = B_i \times h_i(t)$. The global volume of the system $V(t)$ is then equal to:

$$V(t) = B_1 \times h_1(t) + B_2 \times h_2(t) \tag{1}$$

This expression serves as gluing invariant for the system. Note that it is of the form $x_p^A = f(x_{p,1}^C, x_{p,2}^C)$, with $x_p^A = V$, $x_{p,i}^C = h_i$ and $f(h_1, h_2) = B_1 \times h_1 + B_2 \times h_2$.

The defined pump behaviour is modelled by a differential equation for the h_i, based on their status. Let In_i and Out_i the status of the input and of the output pump respectively (with $In_i = 1$ if the pump is open and 0 otherwise); the variation of liquid height in the tank is expressed as:

$$\Delta_i = In_i \times \delta_i^{in} - Out_i \times \delta_i^{out} \tag{2}$$

Concretely, the variation of the liquid height is equal to liquid input minus liquid output from the tank. Using this expression, we derive the tanks' ODE:

$$\Phi_i(t, h) = \Delta_i = In_i \times \delta_i^{in} - Out_i \times \delta_i^{out}$$

Note that, in terms of pattern application, we have $S^A = S_1^C = S_2^C = \mathbb{R}$.

Controller description. Apart from the fact that volume is not directly accessible, the controller for this system remains the same as for the abstract one. It operates in the same modes, which are triggered in the same way, replacing V by $B_1h_1 + B_2h_2$, following the system's gluing invariant.

Additionally, as the behaviour of the pumps is known, we define a set of rules that associate the modes of the controller to the state of the pumps:

- *emptying* mode: input pumps closed ($In_i = 0$), output pumps open ($Out_i = 1$);
- *filling* mode: input pumps open ($In_i = 1$), output pumps closed ($Out_i = 0$);
- *stable* mode: every pump closed ($In_i = Out_i = 0$);
- *normal* mode: pumps are closed or open.

This behaviour is captured and formalised by the *TankModeChange* predicate, defined in the *Flow* theory (see Listing 6):

$$\begin{aligned} &\mathit{TankModeChange}(x_s, (\mathit{In}_1, \mathit{Out}_1), (\mathit{In}_2, \mathit{Out}_2)) \Leftrightarrow \\ &\qquad (x_s = \mathit{emptying} \Rightarrow \mathit{In}_1 = 0 \land \mathit{Out}_1 = 1 \land \mathit{In}_2 = 0 \land \mathit{Out}_2 = 1) \\ &\qquad \land (x_s = \mathit{filling} \Rightarrow \mathit{In}_1 = 1 \land \mathit{Out}_1 = 0 \land \mathit{In}_2 = 1 \land \mathit{Out}_2 = 0) \\ &\qquad \land \ldots \end{aligned} \tag{3}$$

This particular predicate, together with the concrete ODE for the tank allows to enforce requirement **FUN1** of the abstract tank, encoded in guard `grd3` of event `ctrl_actuate_pumps` of the `AbstractTank` model (Listing 10). Concretely, in *emptying* mode, the differential equation yields decreasing solutions, in *filling* mode, the differential equation yields increasing solutions, etc.

Note that it is perfectly possible to define other richer behaviours; the difficulty is then to enforce requirement **FUN1**, or in other words to establish guard strengthening, that preserves safe system behaviour, of `grd3` of the abstract event `ctrl_actuate_pumps`.

Requirements. The requirements of the abstract tank are refined and constrained. In particular, the following items are added:

FUN2 The plant consists of 2 tanks with independent pumps that operate with a constant flow;

FUN3 The controller can only sense the height of liquid in each tank; it computes the volume using the sensed height and the parameters of the tanks;

ENV2 Both tanks have a given bases B_1 and B_2, and a given maximum height $H_{1,max}$ and $H_{2,max}$; the maximal global volume is then $\hat{V}_{max} = B_1 H_{1,max} + B_2 H_{2,max}$.

Refinement feasibility. The added requirement **ENV2** implies that the concrete tanks may not have the same total maximum volume as the abstract tank. This means in particular that there may exist some combination of tanks that violate requirements **SAF1** of the abstract tank.

To avoid this situation and enforce correct refinement, it is required that the total maximum volume of the tanks $\hat{V}_{max}$ is greater than the lower bound for the volume V_{low}: $\hat{V}_{max} \geq V_{low}$. Without this constraint, the system can never enforce invariant **SAF1** at all.

Similarly, the gluing invariant allows to deduce $\dot{V}(t) = B_1 \dot{h}_1(t) + B_2 \dot{h}_2(t)$, and the ODE for the tanks gives the following expression for $\dot{V}$: $\dot{V}(t) = B_1 \Delta_1 + B_2 \Delta_2$.

By rewriting this expression, we observe that: $|\dot{V}(t)| \leq B_1 \cdot \max(\delta_1^{in}, \delta_1^{out}) + B_2 \cdot \max(\delta_2^{in}, \delta_2^{out})$. It follows that **SAF2** is equivalent to the condition $B_1 \cdot \max(\delta_1^{in}, \delta_1^{out}) + B_2 \cdot \max(\delta_2^{in}, \delta_2^{out}) \leq \Delta V_{max}$ where every symbol is constant.

Note that refinement works by strengthening constraints, meaning that we can establish a correct refinement relationship between the abstract and the concrete tanks where the concrete maximum volume is lower than the higher bound (i.e. $V_{low} \leq \hat{V}_{max} < V_{high}$). In this peculiar situation, half of the invariant is statically enforced ($\forall t, \hat{V}(t) \leq V_{high}$), but the system *looses some capabilities*: there exist potential values of the abstract state that can never be reached by the concrete state.

This issue is tied to the meaning we give to the abstract model. On the one hand, it may be seen as a *bound* for the system, i.e. a minimal safety requirement, in which case constraining it is acceptable. On the other hand, it may also be seen as a *correctness* requirement, meaning that the system must not only comply with the given safety bounds but also ensure the given bounds can be reached. In this case, additional invariant properties related to the reachability of all abstract states must be added to the model.

6.3.2. Event-B Development

Context and axioms. Listing 15 gives the context for the designed system. It encompasses the various constants presented during the preliminary study and their associated properties. In particular, `axm13` and `axm23` encode the properties discussed in the requirement section of the preliminary study, regarding the feasibility of this refinement.

Moreover, `axm14` links the initial value for V (V_0) to the initial values of h_1 and h_2. This is so that we can establish simulation for the initialisation in the machine.

Note that additional useful parameters are found in the tank theories (see Section 4.2).

CONTEXT 1Ctrl_2Tanks_Ctx **EXTENDS** AbstractTankCtx
CONSTANTS B_1 , B_2 , $H_{1,max}$, $H_{2,max}$, h_1^0 , h_2^0 ,
δ_1^{in} , δ_1^{out} , δ_2^{in} , δ_2^{out}
AXIOMS
axm1−4 : $B_1, B_2 \in \mathbb{R}$, $0 < B_1$, $0 < B_2$
axm5−8 : $H_{1,max}, H_{2,max}, h_1^0, h_2^0 \in \mathbb{R}$
axm9−10 : $0 < H_{1,max}$, $0 < H_{2,max}$
axm11−12 : $0 < h_1^0 < H_{1,max}$, $0 < h_2^0 < H_{2,max}$
axm13 : $V_{high} \leq B_1 H_{1,max} + B_2 H_{2,max}$
axm14 : $V_0 = B_1 h_1^0 + B_2 h_2^0$
axm15−18 : $\delta_1^{in}, \delta_1^{out}, \delta_2^{in}, \delta_2^{out} \in \mathbb{R}$
axm19−22 : $\delta_1^{in} > 0$, $\delta_1^{out} > 0$, $\delta_2^{in} > 0$, $\delta_2^{out} > 0$
axm23 : $B_1 \max(\delta_1^{in}, \delta_1^{out}) + B_2 \max(\delta_2^{in}, \delta_2^{out}) \leq \Delta V_{max}$
END

Listing 15: 1 Controller 2 Tanks – Context

MACHINE 1Ctrl_2Tanks **REFINES** AbstractTank
SEES 1Ctrl_2Tanks_Ctx
VARIABLES t , x_s , h_1 , h_2
INVARIANTS
inv1 : $h_1 \in \mathbb{R} \nrightarrow \mathbb{R}$
inv2 : $[0, t] \subseteq \text{dom}(h_1)$
inv3 : $h_2 \in \mathbb{R} \nrightarrow \mathbb{R}$
inv4 : $[0, t] \subseteq \text{dom}(h_2)$
inv5 : $V = B_1 h_1 + B_2 h_2$

INITIALISATION
WITH
V' : $V' = B_1 h_1' + B_2 h_2'$
THEN
act1 : $t := 0$
act2 : $x_s := Stable$
act3 : $h1 := \{0 \mapsto h_1^0\}$
act4 : $h2 := \{0 \mapsto h_2^0\}$
END

Listing 16: 1 Controller 2 Tanks – Machine Header

Machine header. The machine's header and initialisation are shown in Listing 16. The machine is, as expected, a refinement of the abstract tank model. Variable V is substituted with variables h_1 and h_2, and the three are linked via the gluing invariant `inv5`, directly taken from Equation 1.

Initialisation is modelled in the same way as for the abstract system, replacing V by the two states h_1 and h_2.

ctrl_transition_normal **REFINES** **ctrl_transition_normal**
WHERE
grd1 : $B_1 h_1(t) + B_2 h_2(t) < V_{high}$
grd2 : $V_{low} < B_1 h_1(t) + B_2 h_2(t)$
THEN
act1 : $x_s := Normal$
END

ctrl_sense_too_high **REFINES** **ctrl_sense_too_high**
WHERE
grd1 : $V_{high} \leq B_1 h_1(t) + B_2 h_2(t)$
THEN
act1 : $x_s := Emptying$
END

Listing 17: Abstract Tank – Transition and Sensing

Discrete events. Listing 17 shows the transition and sensing refined events of the abstract tank model. This refinement follows the principle given in this section. It relies on substituting V with $B_1 h_1 + B_2 h_2$ using the gluing invariant.

Continuous events. Listing 18 shows the system's actuation. It is built following the method given in this section: abstract plant state V is replaced using the gluing invariant, and continuous predicates are updated in order to handle h_1 and h_2 (using two separate ODE) while retaining the exact same evolution domain.

A witness is provided for *eq* to establish event simulation. Note that the state of each valve (io_1 and io_2) of the two tanks are deduced from the system's state (s) using the *TankModeChange* predicate in `grd2` of Listing 18, as defined in the preliminary study for this development (Section 6.3.1).

EVENT **ctrl_actuate_pumps** **REFINES** **ctrl_actuate_pumps**
ANY io_1 , io_2 , s , t'
WHERE
grd0 : $t' \in \mathbb{R} \land t < t'$
grd1 : **Feasible**($[t, t']$, $h_1 \otimes h_2$, $\{\hat{h}_1 \otimes \hat{h}_2, \hat{h}'_1 \otimes \hat{h}'_2 \mid$
solutionOf($[t, t']$, $\hat{h}'_1$, **ode**($FlowODE(0, H_{1,max}, \delta_1^{in}, \delta_1^{out}, io_1), h_1(t), t)$)$\land$
solutionOf($[t, t']$, $\hat{h}'_2$, **ode**($FlowODE(0, H_{2,max}, \delta_2^{in}, \delta_2^{out}, io_2), h_2(t), t)$)$\}$,
$\{h_1^*, h_2^* \mid V_{low} < B_1h_1 + B_2h_2 \land B_1h_1 + B_2h_2 < V_{high}\}$)
grd2 : $TankModeChange(s, io_1, io_2)$
grd4 : $s \in STATES$
grd5 : $x_s = s$
grd6 : $V_{low} \leq B_1h_1(t) + B_2h_2(t) \land B_1h_1(t) + B_2h_2(t) \leq V_{high}$
WITH
V' : $V' = B_1h'_1 + B_2h'_2$
eq : **Solvable**($[t, t']$, eq, $\{V^* \mid V_{low} < V^* \land V^* < V_{high}\}$) $\land$ $isFlowEq(s, [t, t'], eq, 0, V_{max})$
$\land$**solutionOf**($[t, t']$, $B_1h'_1 + B_2h'_2$, eq)
THEN
act1 : $h_1, h_2 :|_{t \to t'}$
solutionOf($[t, t']$, h'_1, **ode**($FlowODE(0, H_{1,max}, \delta_1^{in}, \delta_1^{out}, io_1), h_1(t), t)$)$\land$
solutionOf($[t, t']$, h'_2, **ode**($FlowODE(0, H_{2,max}, \delta_2^{in}, \delta_2^{out}, io_2), h_2(t), t)$)
$\&\{h_1^*, h_2^* \mid V_{low} < B_1h_1 + B_2h_2 \land B_1h_1 + B_2h_2 < V_{high}\}$
END

Listing 18: 1 Controller 2 Tanks – Actuation Event

6.3.3. Proofs

This model is associated with 70 proof obligations. They are mostly related to well-definedness (36%), because of the extensive use of theories in the model (both for continuous features and for the domain-specific tank features). POs relating to invariants (30%) are proven using the various results defined in the tank domain theories, and the remaining POs mainly come from *refinement* (28%) and *feasibility* (6%).

Guard strengthening is immediate, following the remarks of Section 6.2.2: it consists in substituting the abstract and the concrete continuous states using the gluing invariants. For other guards that do not use continuous state, the concrete model defines guards that are stricter than the guards of the abstract model. Guard strengthening proofs are then proven by substituting parameters and variables using the provided witnesses and gluing invariants, and by using the theorems defined in the theory of tanks.

According to Section 6.2.3, we notice that the simulation proof obligations (associated with the actuation event) boil down to establishing *continuous predicate simulation* (*CPSIM*). Unfolding the associated equation and filling its parameters, we obtain the following equation:

$$\begin{aligned} A \land I \land G \land V' \underset{[0,t']}{=} B_1h'_1 + B_2h'_2 \land \mathbf{solutionOf}([t, t'], h'_1, \mathbf{ode}(\Phi_1)) \land \\ \mathbf{solutionOf}([t, t'], h'_2, \mathbf{ode}(\Phi_2)) \land \mathbf{solutionOf}([t, t'], B_1h'_1 + B_2h'_2, eq) \\ \Rightarrow \mathbf{solutionOf}([t, t'], V', eq) \end{aligned} \tag{4}$$

The witness provided for eq (**solutionOf**($[t, t']$, $B_1h'_1 + B_2h'_2$, eq)) constrains the equations of the concrete tanks to be "compatible" with the equations of the abstract tank. This allows to discharge the PO by substituting V' with $B_1h'_1 + B_2h'_2$ (using the witness for V').

The expression of this witness is associated to a feasibility proof obligation: we are required to prove that there exists eq such that Equation 4 holds.

7. Many-to-Many Architectural Pattern

The *many-to-many* (M2M for short) architectural pattern corresponds to scenarios such as the one presented in Figure 24. These kinds of systems consist of several communicating controllers that are linked together via a network (internet, radio or other wireless means, etc.), each of which controls a plant.

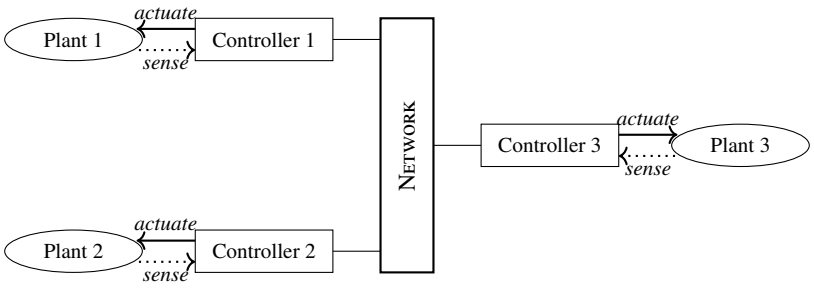

Figure 24: Many-to-many Pattern Typical Scenario

Compared to *single-to-many* systems where one controller is constantly aware of the state of every plant, in a *many-to-many* architecture, controllers only have direct access to their associated plant. Information on the other plants' state is supplied by other controllers, through the communication network. Such exchanges may generate imprecision (e.g. because of communication delays).

This imprecision must be taken into account when designing the system: controllers may have to take decisions depending on a global state, but this imprecision makes it difficult to establish such a global state accurately. This means that controllers must consider and anticipate potential incorrectness in the calculated state, and further constrain their behaviour.

7.1. Pattern Model

Like for the *single-to-many* pattern, we give the M2M pattern as a refinement. Similarly, we assume that the concrete machine only handles two continuous variables. Note that it can easily be extended to handle larger systems.

7.1.1. Machine States and Working Hypotheses

Let M^A be a machine with discrete state $x_s^A \in \text{STATES}^A$ and continuous state $x_p^A \in \mathbb{R} \nrightarrow S^A$. Let M^C be a refinement of M^A consisting of two sub-components, i.e. two controllers, each being linked to one plant. The sub-components are modelled using two discrete states $x_{s,1}^C \in \text{STATES}_1^C$ and $x_{s,2}^C \in \text{STATES}_2^C$ and two continuous states $x_{p,1}^C \in \mathbb{R} \nrightarrow S_1^C$ and $x_{p,2}^C \in \mathbb{R} \nrightarrow S_2^C$.

Gluing invariant. Similarly to the S2M pattern, the gluing invariant between x_p^A and $(x_{p,1}^C, x_{p,2}^C)$ is of the form: $x_p^A = f(x_{p,1}^C, x_{p,2}^C)$, with $f \in S_1^C \times S_2^C \to S^A$.

Global continuous state estimation. In *single-to-single* systems, the controller has direct access to the plant's state (sensing is instantaneous). In the case of *many-to-many* systems, each controller is directly linked to one plant, but cannot sense the other plant's state directly and instantaneously. Instead, each controller sends its associated plant's state to the other controllers through a network, and retrieves through this network the state of the other parts of the system.

This exchange unavoidably introduces imprecision, due to communication delay, rounding errors and so on. Consequently, controllers cannot access an exact state of the global system. Therefore, in order to maintain a global invariant on the system, each controller has to record a safe estimation of the state of the other components.

In practice, this is modelled using specific variables, $x_{p,i}^{sim}$, representing the estimation of the state of the remainder of the system. This state is assumed to be 1) regularly updated, such that the delay between its current value and the actual value of this state is *bounded* (i.e. bounded delay in the communication) 2) the controller is able to *emulate* the behaviour of this variable between two updates, meaning it is able to estimate the behaviour of the remainder of the system, using physics simulation, for example.

Concretely, Controller 1 is keeping track of variable $x_{p,2}^{sim} \in \mathbb{R} \nrightarrow S_2^C$ that simulates Plant 2, and Controller 2 is keeping track of variable $x_{p,1}^{sim} \in \mathbb{R} \nrightarrow S_1^C$ that emulates Plant 1.

Since communication delay is bounded, we assume that the difference between the estimation of the remainder of the state $x^{sim}_{p,i}$ and its actual value $x^{C}_{p,i}$ is bounded. We note Δ^{sim}_{i} this bound, and we have:

$$\forall i \in \{1,2\}, \forall t^* \in [0,t], \|x^{C}_{p,i}(t^*) - x^{sim}_{p,i}(t^*)\| \leq \Delta^{sim}_{i}$$

where $\|\cdot\|$ denotes a norm on the state space S^{C}_{i}.

This bound depends on the specific properties of the network and of the physical phenomena models of the plants. It is usually provided in a specific theory of domains, formalising the application's domain knowledge.

Discrete state. Each controller of the concrete model has its own discrete state. This controller decides how its associated plant is controlled. However, at the abstract level, the defined abstract model describes a single discrete state only, the global state of the system. To ensure the consistency of the refinement of the abstract model, it is required to provide a gluing invariant, linking this abstract discrete global state to the concrete "local" discrete states, that correspond to the discrete state of each controller.

The relationship linking the local discrete states of each controller and global discrete state of the abstract controller is specific to each studied system; it defines how the local controllers behave, depending on the state of each other local controller.

In the following, this relationship is defined as a *policy*. It is modelled using a particular set predicate of possible relations between abstract and concrete discrete states. It is available in a given context or in a domain theory. The policy predicate of the form $(x^{A}_{s}, x^{C}_{s,1}, x^{C}_{s,2}) \in \mathit{Policy}$ establishes this relation.

It is worth noticing that the concept of *policy* arises with the distributed, multi-controller nature of the system, encoded by the two distinct concrete discrete variables $x^{C}_{s,1}$ and $x^{C}_{s,2}$. It is a way for the controllers to "agree" on a set of valid behaviours they shall follow. The definition of this predicate does not prevent from the checking of the refinement consistency as shown below.

Note that, policy is not explicitly defined in the *single-to-many* architectural pattern. Indeed, there exists only a single controller (equivalently, the policy predicate for a centralised controller is the identity: $x^{C}_{s,1} = x^{C}_{s,2} = x^{A}_{s}$).

7.1.2. Machine Header

MACHINE M^A
VARIABLES t, x^A_s, x^A_p
INVARIANTS
inv1 : $t \in \mathbb{R}^+$
inv2 : $x^A_s \in \text{STATES}^A$
inv3 : $x^A_p \in \mathbb{R} \nrightarrow S^A$
inv4 : $[0,t] \subseteq \text{dom}(x^A_p)$

MACHINE M^C **REFINES** M^A
VARIABLES t, $x^C_{s,1}$, $x^C_{s,2}$, $x^C_{p,1}$, $x^C_{p,2}$, $x^{sim}_{p,1}$, $x^{sim}_{p,2}$
INVARIANTS
inv21−22 : $x^C_{s,1} \in \text{STATES}^C$, $x^C_{s,2} \in \text{STATES}^C$
inv31−32 : $x^C_{p,1} \in \mathbb{R} \nrightarrow S^C_1$, $x^C_{p,2} \in \mathbb{R} \nrightarrow S^C_2$
inv41−42 : $[0,t] \subseteq \text{dom}(x^C_{p,1})$, $[0,t] \subseteq \text{dom}(x^C_{p,2})$
inv51−52 : $x^{sim}_{p,1} \in \mathbb{R} \nrightarrow S^C_1$, $x^{sim}_{p,2} \in \mathbb{R} \nrightarrow S^C_2$
inv61−62 : $[0,t] \subseteq \text{dom}(x^{sim}_{p,1})$, $[0,t] \subseteq \text{dom}(x^{sim}_{p,2})$
inv7 : $\forall t^* \cdot t^* \in [0,t] \Rightarrow \|x^C_{p,1}(t^*) - x^{sim}_{p,1}(t^*)\| \leq \Delta^{sim}_1$
$\wedge \|x^C_{p,2}(t^*) - x^{sim}_{p,2}(t^*)\| \leq \Delta^{sim}_2$
inv8 : $(x^A_s, x^C_{s,1}, x^C_{s,2}) \in \mathit{Policy}$
inv9 : $x^A_p = f(x^C_{p,1}, x^C_{p,2})$

Listing 19: M2M Pattern – Machine Header

Listing 19 gives the header of both abstract and refined machines. The main point here is the correct definition of every variable used by the models, namely discrete ($x^{A/C}_{s}$) and continuous ($x^{A/C}_{p}$) states, as well as the variables used for emulating the global state (x^{sim}_{p}).

Following the discussion of Section 7.1.1, the behaviour of the variables is constrained using invariants. Invariant `inv7` ensures that the emulated states do not drift too far from the actual states, and invariant `inv9` is the gluing invariant linking the abstract and concrete continuous states. Invariant `inv8` links the abstract and the concrete discrete states, using the *Policy* operator discussed above.

Initialisation for the machines is given in Listing 20. Variables are initialised as usual; note that we have chosen the simulating variables ($x^{sim}_{p,i}$) to be equal to the exact ones.

INITIALISATION^A
THEN
act1 : $t := 0$
act2 : $x_s^A :\in \text{STATES}^A$
act3 : $x_p^A :\in \{0\} \to S^A$
END

INITIALISATION^C REFINES INITIALISATION^A
WITH
$x_s^{A\prime}$: $(x_s^{A\prime}, x_{p,1}^{C\prime}, x_{p,2}^{C\prime}) \in Policy$
$x_p^{A\prime}$: $x_p^{A\prime} = f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime})$
THEN
act1 : $t := 0$
act2 : $x_{s,1}^C, x_{s,2}^C :\in \text{STATES}^C$
act31 : $x_{p,1}^C, x_{p,1}^{sim} :| x_{p,1}^C\{0\} \to S_1^C \wedge x_{p,1}^{sim} = x_{p,1}^{C\prime}$
act32 : $x_{p,2}^C, x_{p,2}^{sim} :| x_{p,2}^C\{0\} \to S_2^C \wedge x_{p,2}^{sim} = x_{p,2}^{C\prime}$
END

Listing 20: M2M Pattern – Initialisation

Because x_p^A and x_s^A disappear at refinement, witnesses have to be provided. They are based on the refinement's gluing invariants (`inv9` and `inv8`, respectively).

7.1.3. Discrete Events

Sense^A
ANY s^A , p^A
WHERE
grd1 : $s^A \in \mathbb{P}1(\text{STATES}^A)$
grd2 : $p^A \in \mathbb{P}(\text{STATES}^A \times \mathbb{R} \times S^A)$
grd3 : $(x_s^A \mapsto t \mapsto x_p^A(t)) \in p^A$
THEN
act1 : $x_s^A :\in s^A$
END

Sense$^C{}_1$ REFINES Sense^A
ANY s_1^C , p_1^C
WITH
$s^A, x_s^{A\prime}$: $x_s^{A\prime} \in s^A \wedge (x_s^{A\prime}, x_{s,1}^{C\prime}, x_{s,2}^C) \in Policy$
WHERE
grd1 : $s_1^C \in \mathbb{P}1(\text{STATES}^C)$
grd2 : $p_1^C \in \mathbb{P}(\text{STATES}^C \times \mathbb{R} \times (S_1^C \times S_2^C))$
grd3 : $(x_{s,1}^C \mapsto t \mapsto (x_{p,1}^C(t) \mapsto x_{p,2}^{sim}(t))) \in p_1^C$
THEN
act1 : $x_{s,1}^C :\in s_1^C$
END

Sense$^C{}_2$ REFINES Sense^A
ANY s_2^C , p_2^C
WITH
$s^A, x_s^{A\prime}$: $x_s^{A\prime} \in s^A \wedge (x_s^{A\prime}, x_{s,1}^C, x_{s,2}^{C\prime}) \in Policy$
WHERE
grd1 : $s_2^C \in \mathbb{P}1(\text{STATES}^C)$
grd2 : $p_2^C \in \mathbb{P}(\text{STATES}^C \times \mathbb{R} \times (S_1^C \times S_2^C))$
grd3 : $(x_{s,2}^C \mapsto t \mapsto (x_{p,1}^{sim}(t) \mapsto x_{p,2}^C(t))) \in p_2^C$
THEN
act1 : $x_{s,2}^C :\in s_2^C$
END

Listing 21: M2M Pattern – Sensing Events

Listing 21 shows the sensing event for the abstract and concrete machines. The key point is that the abstract sensing event is "split" in two events, one for each controller ($Sense^C{}_1$ and $Sense^C{}_2$).

The main difficulty is to establish a correct relation between abstract and concrete discrete states. This is achieved using an adequate policy, that is used to express the witness for $x_s^{A\prime}$ and s^A. Correct values must be provided for $p_{1/2}^C$ to substitute p^A. In particular, these concrete events are required to *strengthen* `grd3`.

Note that the transition event follows the same pattern, without the sensing predicate p (i.e. `grd2` is removed).

7.1.4. Continuous Events

Listing 22 shows the actuation event for both systems. This actuation updates both the continuous state ($x_{p,1}^C$ and $x_{p,2}^C$) and the emulated state variable of each controller ($x_{p,1}^{sim}$ and $x_{p,2}^{sim}$).

Actuate^A
ANY $\mathcal{P}^A$, s , H^A , t'
WHERE
grd0 : $t' > t$
grd1 : $\mathcal{P}^A \in (\mathbb{R}^+ \nrightarrow S^A) \times (\mathbb{R}^+ \nrightarrow S^A)$
grd2 : **Feasible**$(x_p^A, [t, t'], \mathcal{P}^A, H^A)$
grd3 : $s \subseteq \text{STATES}$
grd4 : $x_s \in s$
grd5 : $H^A \subseteq S^A$
grd6 : $x_p^A(t) \in H^A$
THEN
act1 : $x_p^A :|_{t \to t'} \mathcal{P}^A(x_p^A, x_p^{A\prime})$ & H^A
END

Actuate^C REFINES Actuate^A
ANY $\mathcal{P}_1^C$, $\mathcal{P}_2^C$, s_1^C , s_2^C , $x_{p,1}^{sim*}$, $x_{p,2}^{sim*}$, H^C , t'
WHERE
grd0 : $t' > t$
grd1 : $\mathcal{P}_1^C \in (\mathbb{R}^+ \nrightarrow (S_1^C \times S_2^C)) \times (\mathbb{R}^+ \nrightarrow (S_1^C \times S_2^C)) \wedge$
$\mathcal{P}_2^C \in (\mathbb{R}^+ \nrightarrow (S_1^C \times S_2^C)) \times (\mathbb{R}^+ \nrightarrow (S_1^C \times S_2^C))$
grd2 : **Feasible**$(x_{p,1}^C \otimes x_{p,2}^{sim*} \otimes x_{p,1}^{sim*} \otimes x_{p,2}^C, [t, t'],$
$\mathcal{P}_1^C \times \mathcal{P}_2^C \cap \{\hat{x}_{p,1}, \hat{x}_{p,2}^{sim}, \hat{x}_{p,1}^{sim}, \hat{x}_{p,2} \mid x_{p,1}^{sim} =_{[t,t']} x_{p,1}^{sim*}, x_{p,2}^{sim} =_{[t,t']} x_{p,2}^{sim*}\},$
$H^C \cap \{\hat{x}_{p,1}, \hat{x}_{p,2}^{sim}, \hat{x}_{p,1}^{sim}, \hat{x}_{p,2} \mid \|\hat{x}_{p,1} - \hat{x}_{p,1}^{sim}\| \leq \Delta_1^{sim} \wedge \|\hat{x}_{p,2} - \hat{x}_{p,2}^{sim}\| \leq \Delta_2^{sim}\})$
grd3 : $s_1^C \subseteq \text{STATES}^C \wedge s_2^C \subseteq \text{STATES}^C$
grd4 : $x_{s,1}^C \in s_1^C \wedge s_{s,2}^C \in s_2^C$
grd5 : $x_1^{sim*} \in \mathbb{R} \nrightarrow S_1^C \wedge [t, t'] \subseteq \text{dom}(x_1^{sim*})$
grd6 : $x_2^{sim*} \in \mathbb{R} \nrightarrow S_2^C \wedge [t, t'] \subseteq \text{dom}(x_2^{sim*})$
grd7 : $H^C \subseteq S_1^C \times S_2^C \times S_1^C \times S_2^C$
grd8 : $x_{p,1}^C(t), x_{p,2}^{sim*}(t), x_{p,1}^{sim*}(t), x_{p,2}^C(t) \in H^C$
WITH
$x_p^{A\prime}$: $x_p^{A\prime} = f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime})$
s^A : $(x_s^A, x_{s,1}^C, x_{s,2}^C) \in \mathit{Policy} \wedge x_s^A \in s^A$
THEN
act1 : $x_{p,1}^C, x_{p,1}^{sim}, x_{p,2}^C, x_{p,1}^{sim} :|_{t \to t'}$
$\mathcal{P}_1^C(x_{p,1}^C \otimes x_{p,2}^{sim}, x_{p,1}^{C\prime} \otimes x_{p,2}^{sim\prime}) \wedge \mathcal{P}_2^C(x_{p,1}^{sim} \otimes x_{p,2}^C, x_{p,1}^{sim\prime} \otimes x_{p,2}^{C\prime})$
$\wedge x_{p,1}^{sim} =_{[t,t']} x_{p,1}^{sim*} \wedge x_{p,2}^{sim} =_{[t,t']} x_{p,2}^{sim*}$
$\&\{\hat{x}_{p,1}^C, \hat{x}_{p,2}^{sim}, \hat{x}_{p,1}^{sim}, \hat{x}_{p,2}^{sim*} \mid$
$\hat{x}_{p,1}^C, \hat{x}_{p,2}^{sim}, \hat{x}_{p,1}^{sim}, \hat{x}_{p,2}^C \in H^C$
$\wedge \|\hat{x}_{p,1}^C - \hat{x}_{p,1}^{sim}\| \leq \Delta_1^{sim} \wedge \|\hat{x}_{p,2}^C - \hat{x}_{p,2}^{sim}\| \leq \Delta_2^{sim}\}$
END

Listing 22: M2M Pattern – Actuation Event

The continuous state is updated with two predicates, $\mathcal{P}_1^C$ and $\mathcal{P}_2^C$, each of which relates to a continuous variable and the simulated continuous variable of the other (i.e. $(x_{p,1}^C, x_{p,2}^{sim})$ and $(x_{p,1}^{sim}, x_{p,2}^C)$), thus effectively representing two separate and independent actuations, although they are executed simultaneously.

On the other hand, the functions associated to the simulated continuous variables are updated directly with functions ($x_{p,1}^{sim*}$ and $x_{p,2}^{sim*}$), which models the simulations managed and calculated by the controllers themselves.

The predicate employed in the CBAP operator as well as in the **Feasible** predicate is the Cartesian product $\mathcal{P}_1^C \times \mathcal{P}_2^C$, intersected with an additional predicate enforcing the assignments of $x_{p,1}^{sim}$ and $x_{p,2}^{sim}$.

Similarly, H^A is substituted with H^C, with the additional constraints that each continuous state variable simulation must remain close by Δ_i^{sim} from the associated state variable's actual behaviour (i.e. $\|x_{p,i}^C - x_{p,i}^{sim}\| \leq \Delta_i^{sim}$). This additional constraint ensures the provided simulation functions are sufficient for the controller to behave correctly.

Witnesses are provided for s^A and $x_p^{A\prime}$, following the machine's gluing invariants. Note that we do not give witnesses for H^A and $\mathcal{P}^A$; this fact is discussed further in Section 7.2.

7.2. Proofs

The *single-to-many* architectural pattern yields multiple POs, mostly related to refinement. It is to be noted, again, that since it is a pattern, there is no general method for discharging all of these POs, since they rely on the specific features of the system.

In this section, we focus on invariant, guard strengthening and simulation POs, and we study how they are handled.

7.2.1. Invariant Preservation

The gluing invariant, linking abstract and concrete state variables, are preserved (PO 7 of Table 2b) with the use of witnesses. The discrete gluing invariant (`inv8` in Listing 19) is proved with the help of witnesses for $x_s^{A\prime}$ and using the *policy* predicate. The continuous gluing invariant (`inv9`) is proved using the witnesses provided for $x_p^{A\prime}$, that are based on the relation $x_p^A = f(x_{p,1}^C, x_{p,2}^C)$.

For the preservation of the other invariants (PO 2 of Table 2a), typing and basic constraint invariants (invariants `inv21-22` through `inv61-62`) are established using well-defined and well-constrained predicates. More importantly, invariant `inv7` on the proximity of $x_{p,i}$ and $x_{p,i}^{sim}$ is enforced by guard `grd2` of the concrete actuation.

In a later refinement, concrete values are provided for $x_{p,i}^{sim}$, and a guard strengthening PO is generated. It ensures that this concrete value satisfies the proximity constraint provided by the **Feasible** predicate (i.e. $\|\hat{x}_{p,i} - \hat{x}_{p,i}^{sim}\| \leq \Delta_i^{sim}$).

The preservation of the other, system-specific invariants again depends completely on the nature of these invariants and of the concrete system's behaviour.

7.2.2. Guard Strengthening

Guard strengthening (see PO 6 in Table 2b) appears in particular in sensing events. It ensures that the concrete sensing can only occur when the abstract sensing is.

In concrete sensing event $\text{Sense}^C{}_1$ for instance, the guard strengthening PO has the form:

$$\begin{aligned} A \wedge I \wedge (x_s^A, x_{s,1}^C, x_{s,2}^C) \in \mathit{Policy} \wedge x_p^A = f(x_{p,1}^C, x_{p,2}^C) \wedge \|x_{p,2}^C - x_{p,2}^{sim}\| \leq \Delta_2^{sim} \\ \wedge (x_{s,1}^C, t, (x_{p,1}^C(t), x_{p,2}^{sim}(t))) \in p_1^C \Rightarrow (x_s^A, t, x_p^A(t)) \in p^A \end{aligned}$$

The difficulty is to devise an updated guard (i.e. a value for p_1^C) that allows to establish guard strengthening. In the particular case where the guard is of the form $x_p^A \in \hat{p}^A$ (and $(x_{p,1}^C, x_{p,2}^{sim}) \in \hat{p}_1^C$), we exploit the shape of the gluing invariant and the properties of $x_{p,2}^{sim}$. We then have to find $\hat{p}_1^C$ such that:

$$\begin{aligned} (x_{p,1}^C, x_{p,2}^{sim}) \in \hat{p}_1^C \Rightarrow \forall x_{p,2}^* \cdot x_{p,2}^* \in S_C^2 \wedge \|x_{p,2}^* - x_{p,2}^{sim}\| \leq \Delta_2^{sim} \\ \Rightarrow f(x_{p,1}^C, x_{p,2}^*) \in \hat{p}^A \end{aligned}$$

Indeed, if we have this property, $x_{p,2}^*$ is substituted with $x_{p,2}^C$, and the gluing invariant is exploited to substitute $f(x_{p,1}^C, x_{p,2}^C)$ with x_p^A. This means that, to avoid missing the sensing of an event, the guard needs to take into account the simulation error bound Δ_2^{sim}.

Note that the constraints induced by the communication network and the continuous behaviours of the plants may entail a value of Δ_i^{sim} for which the only correct value for $\hat{p}_C$ is $\emptyset$, i.e. the only correct guard for the concrete event with regard to guard strengthening is $\bot$. In this case, the system loses features (some concrete events are disabled).

For instance, if Δ_i^{sim} is too big, it may be impossible to find a predicate $\hat{p}^C$ strong enough to imply $\hat{p}^A$.

7.2.3. Simulation

As for the *single-to-many* pattern, this pattern generates simulation POs (see PO 5 in Table 2b), in particular associated with continuous events. A simulation proof obligation for the system's actuation (with unfolded CBAP) is written as:

$$\begin{aligned} A \wedge I \wedge G \wedge x_p^{A\prime} \underset{[0,t']}{=} f(x_{p,1}^{C\prime}, x_{p,2}^{C\prime}) \wedge \mathcal{P}_1^C(x_{p,1}^C \otimes x_{p,2}^{sim}, x_{p,1}^{C\prime} \otimes x_{p,2}^{sim\prime}) \wedge \mathcal{P}_2^C(x_{p,1}^{sim} \otimes x_{p,2}^C, x_{p,1}^{sim\prime} \otimes x_{p,2}^{C\prime}) \wedge \\ \forall t^* \cdot t^* \in [0, t'] \Rightarrow \|x_{p,1}^{C\prime}(t^*) - x_{p,1}^{sim\prime}(t^*)\| \leq \Delta_1^{sim} \wedge \|x_{p,2}^{C\prime}(t^*) - x_{p,2}^{sim\prime}(t^*)\| \leq \Delta_2^{sim} \\ \wedge x_{p,1}^{C\prime}(t^*), x_{p,2}^{sim\prime}(t^*), x_{p,1}^{sim\prime}(t^*), x_{p,2}^{C\prime}(t^*) \in H^C \\ \Rightarrow \mathcal{P}^A(x_p^A, x_p^{A\prime}) \wedge \forall t^* \cdot t^* \in [t, t'] \Rightarrow x_p^{A\prime}(t^*) \in H^A \end{aligned}$$

This PO drives the design of $\mathcal{P}_i^C$ and H^C, in a way similar to the *single-to-many* pattern, with the additional inclusion of the simulation error bound. Just like for guard strengthening, the idea is that these predicates shall be strict enough such that, even considering the possible gap between simulated and direct variables, simulation still holds.

Note that, for H^C, the same technique as for guard strengthening is used.

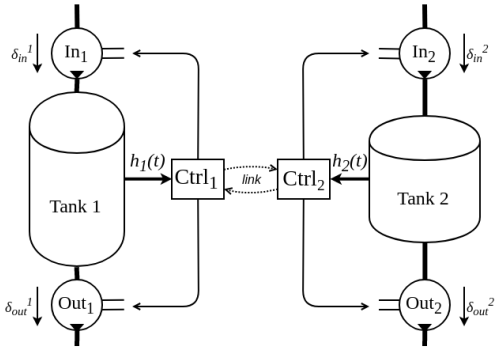

Figure 29: 2 Controllers, 2 Tanks Configuration

7.3. Application to the Case Study

In this section, we propose another refinement of the abstract tank, corresponding to the *many-to-many* architectural pattern presented in this section. It models a system consisting of two tanks of definite shape, each controlled by an independent controller. Figure 29 gives the typical configuration of the system to be designed.

The core of the system is close to the S2M (1 controller 2 tanks) system designed in Section 6.3. It consists of two cylinder tanks with known bases and maximum heights, and the controller accesses the height of liquid in the tank rather than the direct volume (that is calculated). The behaviour of the pumps is identical to that of the S2M case study.

Contrary to the S2M case study where one controller controls several plants (centralised control with the capability to build a global state of the controlled plants), the goal of this refinement is to model a situation where each tank is equipped with a controller, and both controllers *communicate* with each other through a communication network with a bounded delay time.

The difficulty of this development is the computation of a global state (ideal case) or of a safe estimation of a global state, despite the latency of the network and the error that it generates; this is the main use of the *many-to-many* architectural pattern with distributed control.

7.3.1. Preliminary Study

Plant description. The plant is exactly the same as for the *single-to-many* case study (see Section 6.3.1: two cylinder tanks of bases B_1 and B_2 and maximum height $H_{1,max}$ and $H_{2,max}$, containing a height of liquid $h_1(t)$ and $h_2(t)$ and connected to pumps of fixed flow ($\delta_{1/2}^{in/out}$). Consequently, the gluing invariant and the equations used in the model are identical to the S2M case.

Note that, in terms of pattern application, we again have $S^A = S_1^C = S_2^C = \mathbb{R}$.

Controller description. The controllers for this system are more complex than for the S2M case study. The system consists of two controllers, each of which is able to control the pumps and sense the height of liquid in one tank.

The controllers communicate with each other and exchange their current (discrete and continuous) states. The communication is delayed by an arbitrary time, that is assumed to be bounded.

In addition to the pumps they control, and in order to maintain a global safety invariant, each controller sees a global state, or a safe approximation of it. For that, it estimates the continuous state of the other controller, basically by *emulating* it using knowledge of its behaviour and the information it gets from the network. Thus, controller 1 keeps track of the emulated height of tank 2 (h_2^{sim}) and similarly controller 2 keeps track of the emulated height of tank 1 (h_1^{sim}).

We assume that, at any given point, the difference between the real height and the simulated height is bounded by a constant, that basically encompasses the simulation errors and the network's delay:

$$|h_1 - h_1^{sim}| \leq \Delta_1^{sim} \wedge |h_2 - h_2^{sim}| \leq \Delta_2^{sim}$$

Note that no additional assumption is made on *how* the controller calculates h_i^{sim}.

Finally, each subsystem has its own discrete state, taken in the **TankState** type (i.e. $STATES_1^C = STATES_2^C = STATES^A =$ **TankState**). These states together with the discrete state of the abstract machine are linked via the *Policy* set, that is left abstract in this development. Note that, in practice, any policy may be proposed, provided it is compatible with the system's invariants. For example, the policy can be defined based on the principle of keeping each tank level of liquid close to the average.

Requirements. The goal of the system is, as for the abstract one, to keep the global volume ($B_1h_1 + B_2h_2$) within the bounds V_{low}, V_{high}. The requirements are enriched to reflect the particular architecture of the system:

ENV3 each tank is controlled independently by communicating controllers; communication introduces bounded delays and errors, encapsulated in a constant, Δ_i^{sim}, for each controller i.

Refinement feasibility. Like for the *single-to-many* tank control case study, the physical properties of the tanks (base and maximum height) may induce a system that cannot be safe, or that may be too constrained, and misses relevant possible behaviours. As discussed in the preliminary study for the *single-to-many* case study (Section 6.3.1), it is required that the total maximum volume of the tanks $\hat{V}_{max}$ is greater than the lower bound for the volume V_{low}, or possibly greater than the higher bound for the volume to avoid loosing capabilities.

7.3.2. Event-B Development

CONTEXT 2Ctrl_2Tanks_Ctx **EXTENDS** AbstractTankCtx
CONSTANTS B_1, B_2, $H_{1,max}$, $H_{2,max}$, h_1^0, h_2^0,
δ_1^{in}, δ_1^{out}, δ_2^{in}, δ_2^{out},
Δ_1^{sim}, Δ_2^{sim},
Policy
AXIOMS
axm1–4: $B_1, B_2 \in \mathbb{R}$, $0 < B_1$, $0 < B_2$
axm5–8: $H_{1,max}, H_{2,max}, h_1^0, h_2^0 \in \mathbb{R}$
axm9–10: $0 < H_{1,max}$, $0 < H_{2,max}$
axm11–12: $0 < h_1^0 < H_{1,max}$, $0 < h_2^0 < H_{2,max}$
axm13: $V_{high} \leq B_1H_{1,max} + B_2H_{2,max}$
axm14: $V_0 = B_1h_1^0 + B_2h_2^0$
axm15–18: $\delta_1^{in}, \delta_1^{out}, \delta_2^{in}, \delta_2^{out} \in \mathbb{R}$
axm19–22: $\delta_1^{in} > 0$, $\delta_1^{out} > 0$, $\delta_2^{in} > 0$, $\delta_2^{out} > 0$
axm23: $B_1 \max(\delta_1^{in}, \delta_1^{out}) + B_2 \max(\delta_2^{in}, \delta_2^{out}) \leq \Delta V_{max}$
axm24–25: $\Delta_1^{sim} \in \mathbb{R}$, $\Delta_2^{sim} \in \mathbb{R}$
axm26–27: $\Delta_1^{sim} > 0$, $\Delta_2^{sim} > 0$
axm28: $Policy \subseteq \text{STATES} \times \text{STATES} \times \text{STATES}$
END

Listing 23: 2 Controllers 2 Tanks – Context

Constants and axioms. The context for this machine is given in Listing 23. It is very much based on the context of the *single-to-many* case study (Listing 15) with the added constants Δ_i^{sim} and *Policy* required by the pattern. It provides an axiomatisation for the set of properties and constraints outlined in the preliminary study of Section 7.3.1. For example `axm13` describes the refinement feasibility condition.

We recall that other operators are found in the tank theories (see Section 4.2).

Machine header. Listing 24 shows the machine's header and initialisation. Note that, as a matter of readability, we often give the invariants of one of the controllers; the others are abbreviated with dots and can be obtained by switching the indexes.

A big part of the header (`inv1-12` and `inv17-18`) is dedicated to define the numerous variables together with their associated types and basic properties. In particular, `inv9-12` ensure that the height of liquid in each tank ($x_{p,i}^C$), as well as their emulated (approximated) counterparts ($x_{p,i}^{sim}$), is bounded by 0 and $H_{i,max}$.

Invariants `inv13-14` encode the properties of the simulated variables (h_i^{sim}), that must not drift away too far from their real counterpart.

MACHINE 2Ctrl_2Tanks **REFINES** AbstractTank
SEES 2Ctrl_2Tanks_Ctx
VARIABLES t , $x^C_{s,1}$, $x^C_{s,2}$, h_1 , h_2 , h_1^{sim} , h_2^{sim}
INVARIANTS
inv1–4 : $h_1, h_2, h_1^{sim}, h_2^{sim} \in \mathbb{R} \nrightarrow \mathbb{R}$
inv5–8 : $[0,t] \subseteq \mathrm{dom}(h_1)$, ...
inv9–12 : $\forall t^* \cdot t^* \in [0,t] \Rightarrow 0 \leq h_1(t^*) \wedge h_1(t^*) \leq H_{1,max}$, ...
inv13–14 : $\forall t^* \cdot t^* \in [0,t]$
$\Rightarrow |h_1(t^*) - h_1^{sim}(t^*)| \leq \Delta_1^{sim}$, ...
inv15–16 : $\forall t^* \cdot t^* \in [0,t]$
$\Rightarrow B_1 h_1(t^*) + B_2 h_2^{sim}(t^*) \geq V_{low} + B_2 \Delta_2^{sim}$
$\wedge B_1 h_1(t^*) + B_2 h_2^{sim}(t^*) \leq V_{high} - B_2 \Delta_2^{sim}$, ...
inv17–18 : $x_{s,1} \in STATES$, $x_{s,2} \in STATES$
inv19 : $V =_{[0,t]} B_1 h_1 + B_2 h_2$
inv20 : $(x_s, x_{s,1}, x_{s,2}) \in Policy$

INITIALISATION
WITH
V' : $V' =_{[0,t']} B_1 h_1 + B_2 h_2$
x'_s : $(x'_s, x'_{s,1}, x'_{s,2}) \in Policy$
THEN
act1 : $t := 0$
act2 : $x_{s,1}, x_{s,2} := Stable, Stable$
act3 : $h_1, h_2 := 0 \mapsto h_1^0, 0 \mapsto h_2^0$
act4 : $h_1^{sim}, h_2^{sim} := 0 \mapsto h_1^0, 0 \mapsto h_2^0$
END

Listing 24: 2 Controllers 2 Tanks – Machine Header

`inv15-16` give an additional interesting properties, built on the principle explained in Section 7.2.2. The idea is that, if the estimated volume, despite the potential error Δ_i^{sim} is in the bounds, then the actual volume is as well.

Finally, `inv19` and `inv20` encode the gluing invariant of this refinement, for the continuous and discrete states, respectively.

In addition to this header, the initialisation is straightforward: both sub-components are in *stable* mode with a given initial amount of liquid inside. The emulated variables receive the same initial value as a starting point (but could also get another value provided it does not violate the invariants). A witness is provided for the substituted variables, following the gluing invariant.

ctrl_transition_normal_1 REFINES ctrl_transition_normal
WHERE
grd1 : $B_1 h_1(t) + B_2 h_2^{sim}(t) < V_{high} - B_2 \Delta_2^{sim}$
grd2 : $V_{low} + B_2 \Delta_2^{sim} < B_1 h_1(t) + B_2 h_2^{sim}(t)$
THEN
act1 : $x_s^1 := Normal$
END

ctrl_transition_normal_2 REFINES ctrl_transition_normal
WHERE
grd1 : $B_1 h_1^{sim}(t) + B_2 h_2(t) < V_{high} - B_1 \Delta_1^{sim}$
grd2 : $V_{low} + B_1 \Delta_1^{sim} < B_1 h_1^{sim}(t) + B_2 h_2(t)$
THEN
act1 : $x_s^2 := Normal$
END

Listing 25: 2 Controllers 2 Tanks – Transition Events

Discrete events. Listing 25 presents two transition events of the machine. Following the pattern, they are duplicates of the abstract `ctrl_transition_normal` transition event, and each event references only the variables of one controller at a time.

Following the remarks on guard strengthening, we need to ensure that, even if the emulated variables are the furthest allowed by their real counterpart, the guard is still at least as strong as the abstract one. Note that:

$$B_1 h_1(t) + B_2 h_2^{sim}(t) \leq V_{high} - B_2 \Delta_2^{sim} \Rightarrow B_1 h_1(t) + B_2 (h_2^{sim}(t) + \Delta_2^{sim}) \leq V_{high}$$

With $|h_2(t) - h_2^{sim}(t)| \leq \Delta_2^{sim}$, we have $h_2^{sim}(t) + \Delta_2^{sim} \geq h_2(t)$ and deduce:

$$B_1 h_1(t) + B_2 h_2(t) \leq B_1 h_1(t) + B_2 (h_2^{sim}(t) + \Delta_2^{sim}) \leq V_{high}$$

We proceed symmetrically for controller 2 and for the other predicate (with V_{low}).

For the record, we give the sensing events of the machine (Listing 26). They are constructed in the same way as the transition events, and following the methodology of the *many-to-many* architectural pattern given in this section.

ctrl_sense_too_high_1 REFINES ctrl_sense_too_high
WHERE
grd1 : $V_{high} - B_2\Delta_2^{sim} \leq B_1 h_1(t) + B_2 h_2^{sim}(t)$
THEN
act1 : $x_s^1 := Emptying$
END

ctrl_sense_too_high_2 REFINES ctrl_sense_too_high
WHERE
grd1 : $V_{high} - B_1\Delta_1^{sim} \leq B_1 h_1^{sim}(t) + B_2 h_2(t)$
THEN
act1 : $x_s^2 := Emptying$
END

Listing 26: 2 Controllers 2 Tanks – Sensing Events

EVENT ctrl_actuate_pumps REFINES ctrl_actuate_pumps
ANY s_1 , s_2 , t' , io_1 , io_2 , h_1^{sim*} , h_2^{sim*}
WHERE
grd0 : $t' \in \mathbb{R} \wedge t < t'$
grd1 : **Feasible**$([t,t'], h_1 \otimes h_2^{sim}, \{\hat{h}_1, \hat{h}_2^{sim}, \hat{h}_1', \hat{h}_2^{sim'} \mid$
solutionOf$(\hat{h}_1', \textbf{ode}(FlowODE(0, H_{1,max}, \delta_1^{in}, \delta_1^{out}, io_1), h_1(t), t))$
$\wedge \hat{h}_2^{sim'} =_{[t,t']} h_2^{sim*}\}$,
$\{\hat{h}_1, \hat{h}_2^{sim} \mid V_{low} + B_2\Delta_2^{sim} \leq B_1\hat{h}_1 + B_2\hat{h}_2^{sim} \wedge V_{high} - B_2\Delta_2^{sim} \geq B_1\hat{h}_1 + B_2\hat{h}_2^{sim}\}$)
grd2 : ... *-- similar to grd1*
grd3 : $s_1, s_2 \in STATES$
grd4 : $x_s^1 = s_1 \wedge x_s^2 = s_2$
grd5 : $(s_1, io_1, io_2) \in Policy$
grd6 : $V_{low} + B_2\Delta_2^{sim} \leq B_1 h_1(t) + B_2 h_2^{sim}(t) \wedge V_{high} - B_1\Delta_1^{sim} \geq B_1 h_1(t) + B_2 h_2^{sim}(t)$
grd7 : $V_{low} + B_1\Delta_1^{sim} \leq B_1 h_1^{sim}(t) + B_2 h_2(t) \wedge V_{high} - B_2\Delta_2^{sim} \geq B_1 h_1^{sim}(t) + B_2 h_2(t)$
WITH
V' : $V' = [0,t']B_1 h_1' + B_2 h_2'$
s : $(s, s_1, s_2) \in Policy$
eq : **Solvable**$([t,t'], eq, \{V^* \mid V_{low} < V^* \wedge V^* < V_{high}\}) \wedge isFlowEq(s, [t,t'], eq, 0, V_{max})$
$\wedge$**solutionOf**$([t,t'], B_1 h_1' + B_2 h_2', eq)$
THEN
act1 : $h_1, h_2, h_1^{sim}, h_2^{sim} :|_{t \to t'}$
solutionOf$([t,t'], h_1', \textbf{ode}(FlowODE(0, H_{1,max}, \delta_1^{in}, \delta_1^{out}, io_1), h_1(t), t)) \wedge$
solutionOf$([t,t'], h_2', \textbf{ode}(FlowODE(0, H_{2,max}, \delta_2^{in}, \delta_2^{out}, io_2), h_2(t), t)) \wedge$
$h_1^{sim'} = h_1^{sim*} \wedge h_2^{sim'} = h_2^{sim*}$
$\&\{\hat{h}_1, \hat{h}_2^{sim}, \hat{h}_1^{sim}, \hat{h}_2 \mid$
$V_{low} + B_2\Delta_2^{sim} \leq B_1\hat{h}_1 + B_2\hat{h}_2^{sim} \wedge V_{high} - B_2\Delta_2^{sim} \geq B_1\hat{h}_1 + B_2\hat{h}_2^{sim} \wedge$
$V_{low} + B_1\Delta_1^{sim} \leq B_1\hat{h}_1^{sim} + B_2\hat{h}_2 \wedge V_{high} - B_1\Delta_1^{sim} \geq B_1\hat{h}_1^{sim} + B_2\hat{h}_2$
$\}$
END

Listing 27: 2 Controllers 2 Tanks – Actuation Event

Continuous events. Listing 27 shows the system's actuation, built from the concepts presented in the *single-to-many* pattern case study, but following the *many-to-many* architectural pattern. The goal of this actuation is to update the behaviour of the two continuous states and two emulated continuous states, according to the current discrete state of each controller.

The evolution domain ensures that the estimated global volume remains within the bound, including the allowed, bounded imprecision.

A witness is given for *eq*, the same as for the *single-to-many* case study, so that we can establish simulation.

Note that the functions associated with the emulated variables are not given explicitly; the way they are calculated could be specified in a later refinement (linear/polynomial approximation, stochastic models, etc.).

7.3.3. Proofs

This model is associated with 153 POs in total, most of which are related to well-definedness (27%) of the use of the operators defined in the theories and to invariants (47%), and in particular typing invariants. The other POs relate to refinement mainly (21%), and to feasibility of actions and witnesses (5%).

Guard strengthening is addressed when presenting the discrete events: by narrowing the bounds, we ensure that, despite the imprecision caused by the estimation of the global state, the tanks are safely controlled.

Simulation of the Event-B machines is ensured by the particular shape of the witnesses and witness feasibility proof obligations are generated.

8. Conclusion

The work presented in this paper formalises different hybrid system architectures in the form of a refinement pattern, allowing to introduce various structures as a development step. The resulting patterns are presented in the form of a refinement, to be applied for specific hybrid systems. These patterns are adaptable and useful, as they may be used at any point during the refinement chain, and can be composed to produce rich and complex architectures provided that the invariants gluing the abstract component and the components resulting from the decomposition hold. In addition, being presented under the form of a refinement, it is possible to use these patterns on a global system, or on one of its sub-components. This makes these patterns composable to incrementally produce rich and complex architectures.

Assessment

The proposed patterns have been applied to the case study of the control of a liquid volume level of a tank. The single-to-many pattern has been successfully applied to the case study of controlling the filling and emptying of two water tanks connected to a centralised controller. The many-to-many pattern has been applied to a similar case study, with two water tanks independently controlled by two controllers, communicating with bounded delay. The complete Event-B models for both cases studies are available at `https://irit.fr/~Guillaume.Dupont/models.php`.

In addition, these case studies also demonstrate the interest of providing domain theories that axiomatise the physics of the plants to be controlled. These *domain theories* encapsulate domain-specific knowledge related to tanks liquid level control, and in particular hypotheses issued from physics, and tanks and valves specific constraints (e.g. communication delay, volume computation according to the shape of the tanks, etc.).

Note that one may consider, as a limitation of our approach, that the presence of only two tanks in our case study means that the presented patterns only allow to refine one abstract component with two concrete components. This is not the case as the presented patterns allow decomposition in more than two components by successive applications of the patterns. However, a direct instantiation of the patterns for an arbitrary number of components (greater than two) can be achieved using additional theories, in particular a theory of vectors.

Last, we mention that the *many-to-many* pattern can be seen as a model of a class of cyber-physical or autonomous systems, or in other words sets of hybrid systems that communicate with each other with decentralised control. However, such models require the availability of a constant to bound the communication delay between the controllers ensuring the whole system remains in a safe state. Computing this bounded delay is out of the scope of this paper.

Pattern-oriented design

Rather than designing an *ad hoc* hybrid system modelling language relying on Event-B we have chosen to rely on the definition of different formalised patterns on the one hand and on other domain theories on the other hand. Indeed, other patterns, like approximation and linearisation, in addition to the ones presented in this paper have been developed. Other domain theories axiomatising specific knowledge domains, in particular the knowledge related to the physics of the plants, have been defined. Examples of such domains are kinematics and oscillation behaviour. The obtained framework is depicted in Figure 35.

Proceeding in this way makes the proposed framework extensible and generic. Extensibility is ensured by the capability to introduce new formalised patterns, linked to the existing ones by a refinement relationship, and new relevant theories that may be required by the introduced patterns.

Use of this framework for the development of specific hybrid systems, requires first to define the domain theories needed to model the domain knowledge related to the specific system and the refinement chain that instantiates the corresponding patterns as shown at the bottom of Figure 35.

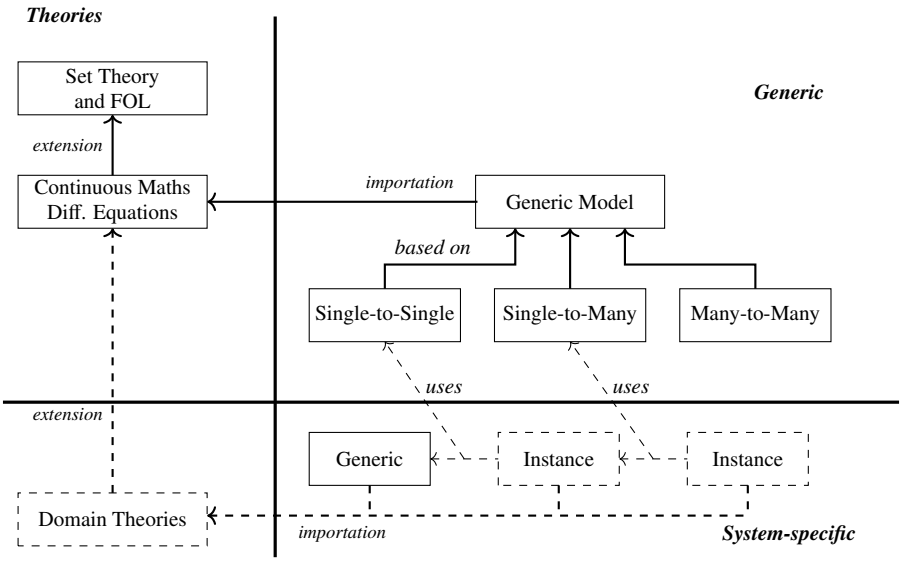

Figure 35: Framework Tools – Architectural Patterns

Perspectives

The work presented in this paper can be pursued in many directions. First, further studies on the many-to-many pattern can be carried out in order to support distributed control to guarantee global invariants deduced from the local invariants of each hybrid system taking part in the whole system. In particular, allowing the capability to add and/or remove a given component i.e. a hybrid system makes it possible to model dynamic systems and a sort of autonomy through this pattern. The definition of the timing analysis that provided the bound for communication is fixed as a constant in the current model. This value is issued from analysis of timing delays in the network of the communicating hybrid systems that is not explicitly described in the developed model.

Additional patterns may be defined in the framework, to handle other development operations, common in the design of hybrid systems. In particular, *discretisation* operations would allow to formally transform a continuous model of a hybrid system into a discrete one, closer to implementation, while ensuring the preservation of its properties. This operation could be further extended to handle floating-point numbers, as an approximated implementation of real numbers on real-world machines. Other types of approximations may be handled, for instance *proportional-integration-derivation* (PID), that addresses specific types of hybrid systems and are widely used in controller design.

Expanding on the idea of patterns and framework [38], it is possible to propose (formal) *software product lines* [39, 40] dedicated to hybrid systems, further embedding design and verification. Event-B theories already make domain features reusable, and additional automations and reusable aspects may be envisioned, in particular on the proving side and on pattern instantiation. For instance, our work initiated in [41, 38] shows how Event-B refinement can be used to support *approximation* of hybrid models. This type of embedding may be used in the context of architectural patterns, to improve their usability, especially on the proof side.

Additional theories able to handle a larger panel of hybrid systems should be defined as well. For instance, in our work, we only handle ordinary differential equations (ODEs), which is the most common type of continuous behaviour models in hybrid systems; but other types of differential equations, together with their relevant properties, may be formalised in theories (e.g. partial differential equations). Regarding the specific hybrid systems, specific domain theories for physics may be defined, e.g. for kinematics, thermodynamics, fluid mechanics, etc. These theories are useful to design application-specific domain theories; for instance, theories for self-driving cars, trains, planes, and so on may be proposed to address several case studies in these respective domains.

Last, we would like to address the aspect of refinement feasibility and potential feature loss mentioned in Section 6.3.1. In fact, the situation we pointed out relates to a general problem of *reachability* on hybrid systems (i.e. both discrete and continuous state reachability). This problem is hard to tackle in Event-B. A *domain-based* approach would allow to give conditions on refinement feasibility for specific types of systems, but can hardly be generalised. Another

possibility is to use hybrid model-checkers, particularly good at reachability analysis, to provide useful information on the consistency of the constraints of the system.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, Theoretical Computer Science 138 (1) (1995) 3–34.

[2] A. Platzer, Differential dynamic logic for hybrid systems, Journal of Automated Reasoning 41 (2) (2008) 143–189.

[3] N. Aréchiga, S. M. Loos, A. Platzer, B. H. Krogh, Using theorem provers to guarantee closed-loop system properties, in: American Control Conference (ACC), 2012, pp. 3573–3580.

[4] P. O. J. Scherer, Discretization of Differential Equations, Springer International Publishing, 2013, pp. 177–205.

[5] R. Banach, M. Butler, S. Qin, N. Verma, H. Zhu, Core Hybrid Event-B I: Single Hybrid Event-B machines, Science of Computer Programming (2015).

[6] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, L. Zou, A calculus for hybrid CSP, in: K. Ueda (Ed.), Programming Languages and Systems - 8th APLAS Symposium, Vol. 6461 of LNCS, Springer, 2010, pp. 1–15.

[7] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.

[8] R.-J. Back, L. Petre, I. Porres, Generalizing action systems to hybrid systems, in: Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT '00, Springer-Verlag, London, UK, 2000, pp. 202–213.

[9] R.-J. Back, R. Kurki-Suonio, Decentralization of process nets with centralized control, Distributed Computing 3 (June 1989). doi:10.1007/BF01558665.

[10] S. Boldo, C. Lelay, G. Melquiond, Coquelicot: A user-friendly library of real analysis for Coq, Mathematics in Computer Science 9 (1) (2015) 41–62.

[11] H. Herencia-Zapana, R. Jobredeaux, S. Owre, P.-L. Garoche, E. Feron, G. Perez, P. Ascariz, PVS linear algebra libraries for verification of control software algorithms in C/ACSL, in: A. E. Goodloe, S. Person (Eds.), NASA Formal Methods, Springer Berlin Heidelberg, 2012, pp. 147–161.

[12] G. Dupont, Y. Aït-Ameur, M. Pantel, N. K. Singh, Proof-based approach to hybrid systems development: Dynamic logic and Event-B, in: M. Butler, A. Raschke, T. S. Hoang, K. Reichl (Eds.), Abstract State Machines, Alloy, B, TLA, VDM, and Z, Springer International Publishing, Cham, 2018, pp. 155–170.

[13] G. Dupont, Y. A. Ameur, N. K. Singh, M. Pantel, Event-B hybridation: A proof and refinement-based framework for modelling hybrid systems, ACM Transaction on Embedded Computer Systems 20 (4) (2021) 35:1–35:37. doi:10.1145/3448270.

[14] G. Dupont, Y. Aït-Ameur, M. Pantel, N. K. Singh, Hybrid systems and Event-B: A formal approach to signalised left-turn assist, in: New Trends in Model and Data Engineering, Springer International Publishing, 2018, pp. 153–158.

[15] P. Stankaitis, G. Dupont, N. K. Singh, Y. Ait-Ameur, A. Iliasov, A. Romanovsky, Modelling hybrid train speed controller using proof and refinement, in: 24th International Conference on Engineering of Complex Computer Systems (ICECCS), 2019, pp. 107–113.

[16] G. Dupont, Y. Aït-Ameur, M. Pantel, N. K. Singh, Handling refinement of continuous behaviors: A refinement and proof based approach with Event-B, in: 13th International Symposium TASE, IEEE Computer Society Press, 2019, pp. 9–16.

[17] G. Dupont, Y. Aït-Ameur, M. Pantel, N. K. Singh, Formally verified architecture patterns of hybrid systems using proof and refinement with Event-B, in: A. Rashke, D. Méry (Eds.), 7th International Conference, ABZ 2020, Proceedings, Vol. 12071 of LNCS, Springer, 2020, pp. 155–170.

[18] S. Lunel, B. Boyer, J.-P. Talpin, Compositional proofs in differential dynamic logic dL, in: 17th International Conference on Application of Concurrency to System Design, ACSD, IEEE Computer Society, 2017, pp. 19–28.

[19] S. Lunel, S. Mitsch, B. Boyer, J.-P. Talpin, Parallel composition and modular verification of computer controlled systems in differential dynamic logic, in: M. H. ter Beek, A. McIver, J. N. Oliveira (Eds.), Formal Methods – The Next 30 Years (FM 2019), Vol. 11800 of LNCS, Springer International Publishing, Cham, 2019, pp. 354–370.

[20] R. Banach, Pliant modalities in Hybrid Event-B, in: Z. Liu, J. Woodcock, H. Zhu (Eds.), Theories of Programming and Formal Methods, Vol. 8051 of LNCS, Springer Berlin Heidelberg, 2013, pp. 37–53.

[21] R. Banach, M. Butler, S. Qin, H. Zhu, Core Hybrid Event-B II: Multiple cooperating Hybrid Event-B machines, Science of Computer Programming 139 (2017) 1 – 35.

[22] H. Jifeng, A classical mind, Prentice Hall International (UK) Ltd., 1994, Ch. From CSP to Hybrid Systems, pp. 171–189.

[23] H. Zhao, M. Yang, N. Zhan, B. Gu, L. Zou, Y. Chen, Formal verification of a descent guidance control program of a lunar lander, in: C. Jones, P. Pihlajasaari, J. Sun (Eds.), FM 2014: Formal Methods, Vol. 8442 of LNCS, Springer International Publishing, Cham, 2014, pp. 733–748.

[24] L. Zou, J. Lv, S. Wang, N. Zhan, T. Tang, L. Yuan, Y. Liu, Verifying Chinese train control system under a combined scenario by theorem proving, in: E. Cohen, A. Rybalchenko (Eds.), Verified Software: Theories, Tools, Experiments (VSTTE 2013), Vol. 8164 of LNCS, Springer Berlin Heidelberg, 2014, pp. 262–280.

[25] S. M. Loos, A. Platzer, Differential refinement logic, in: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, ACM, New York, NY, USA, 2016, pp. 505–514.

[26] J.-D. Quesel, S. Mitsch, S. Loos, N. Aréchiga, A. Platzer, How to model and prove hybrid systems with KeYmaera: a tutorial on safety, International Journal on Software Tools for Technology Transfer 18 (1) (2016) 67–91.

[27] A. Platzer, J.-D. Quesel, European train control system: A case study in formal verification, in: Formal Methods and Software Engineering: 11th International Conference on Formal Engineering Methods (ICFEM 2009), Rio de Janeiro, Brazil., Vol. 5885 of LNCS, Springer Berlin Heidelberg, 2009, pp. 246–265.

[28] A. Platzer, E. M. Clarke, Formal verification of curved flight collision avoidance maneuvers: A case study, in: Formal Methods: Second World Congress (FM 2009), Eindhoven, The Netherlands, November 2-6, 2009. Proceedings, Vol. 5850 of LNCS, Springer Berlin Heidelberg, 2009, pp. 547–562.

[29] Y. Kouskoulas, D. Renshaw, A. Platzer, P. Kazanzides, Certifying the safe design of a virtual fixture control algorithm for a surgical robot, in: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, HSCC '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 263–272.

[30] R. Banach, Formal refinement and partitioning of a fuel pump system for small aircraft in Hybrid Event-B, in: 10th International Symposium on Theoretical Aspects of Software Engineering (TASE), 2016, pp. 65–72.

[31] R. Banach, Hemodialysis machine in hybrid Event-B, in: M. Butler, K.-D. Schewe, A. Mashkoor, M. Biro (Eds.), Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2016), Vol. 9675 of LNCS, Springer International Publishing, Cham, 2016, pp. 376–393.

[32] J.-R. Abrial, Modeling in Event-B: System and Software Engineering, 1st Edition, Cambridge University Press, New York, NY, USA, 2010.

[33] M. Butler, I. Maamria, Practical theory extension in Event-B, in: Z. Liu, J. Woodcock, H. Zhu (Eds.), Theories of Programming and Formal Methods, Vol. 8051 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 67–81.

[34] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoàng, F. Mehta, L. Voisin, Rodin: an open toolset for modelling and reasoning in Event-B, International Journal Software Tools for Technology Transfer 12 (6) (2010) 447–466.

[35] J.-R. Abrial, M. Butler, S. Hallerstede, M. Leuschel, M. Schmalz, L. Voisin, Proposals for mathematical extensions for Event-B, Tech. rep. (2009).

[36] G. Dupont, Correct-by-construction design of hybrid systems based on refinement and proof, Ph.D. thesis, Institut National Polytechnique de Toulouse, Toulouse, France (2021).
URL `https://irit.fr/~Guillaume.Dupont/thesis/Thesis.pdf`

[37] L. Meinicke, I. J. Hayes, Continuous action system refinement, in: Proceedings of the 8th International Conference on Mathematics of Program Construction (MPC 06), Vol. 4014 of LNCS, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 316–337.

[38] G. Dupont, Y. Aït-Ameur, M. Pantel, N. K. Singh, An Event-B based generic framework for hybrid systems formal modelling, in: C. A. Furia, B. Dongol, E. Troubitsyna (Eds.), 16th International Conference on integrated Formal Methods, iFM 2020, Vol. 12546 of LNCS, Springer, 2020, pp. 82–102.

[39] M. H. ter Beek, E. P. de Vink, Towards modular verification of software product lines with mCRL2, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, Springer Berlin Heidelberg, 2014, pp. 368–385.

[40] R. Muschevici, J. Proença, D. Clarke, Modular modelling of software product lines with feature nets, in: G. Barthe, A. Pardo, G. Schneider (Eds.), 9th International Conference on Software Engineering and Formal Methods, SEFM 2011, Springer Berlin Heidelberg, 2011, pp. 318–333.

[41] G. Dupont, Y. Aït-Ameur, N. K. Singh, F. Ishikawa, T. Kobayashi, M. Pantel, Embedding approximation in Event-B: Safe hybrid system design using proof and refinement, in: J. S. Dong, J. McCarthy (Eds.), 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Vol. 12531 of LNCS, Springer, 2020, pp. 251–267.