A method for rigorous design of reconfigurable systems[☆]

Alexandre Madeira[a], Renato Neves[a], Luís S. Barbosa[a], Manuel A. Martins[b]

[a] *INESC TEC and Universidade do Minho, Portugal*
[b] *CIDMA and Department of Mathematics, Universidade de Aveiro, Portugal*

Abstract

Reconfigurability, understood as the ability of a system to behave differently in different modes of operation and commute between them along its lifetime, is a cross-cutting concern in modern Software Engineering. This paper introduces a specification method for reconfigurable software based on a global transition structure to capture the system's reconfiguration space, and a local specification of each operation mode in whatever logic (equational, first-order, partial, fuzzy, probabilistic, etc.) is found expressive enough for handling its requirements.

In the method these two levels are not only made explicit and juxtaposed, but formally interrelated. The key to achieve such a goal is a systematic process of hybridisation of logics through which the relationship between the local and global levels of a specification becomes internalised in the logic itself.

Keywords: Software specification, reconfigurable systems, hybrid logic

1. Introduction

1.1. Motivation: The insulin infusion pump

Diabetes mellitus is one of the most rapidly growing diseases in modern times, predicted by many to become one of the most challenging health problems of this century. There is, currently, no known cure for diabetes. Moreover, in its most severe forms, the standard treatment still boils down to insulin injections, a method that hinders a patient's normal life and carries risks. For this reason, new, more convenient, less expensive and safer technologies helping patients to pursue a better quality of life, are in order.

The insulin infusion pump (IIP) is an alternative to the usual treatment. To compensate for eventual insulin disequilibriums the IIP measures at regular intervals the patient needs and injects insulin through a subcutaneous catheter at different rates. As an alternative to insulin injections, it brings greater dosing precision and quicker adjustments on the fly.

The operation of a IIP is, of course safety critical in the sense that a small error in, for example, administrating the correct dose of insulin, may cause severe harm, or even death. Potential risks are well-studied (*cf.* [112, 111]), but can be mitigated through rigorous design, able to enforce implementations to obey to (suitably formalised) safety requirements. Not surprisingly the design of a safe IIP is regarded as a challenge in the engineering of safety critical systems. Specific strategies have been proposed resorting to, and even combining, different formalisms. Among several others one may cite the work reported in [7], in which state-based specifications are validated in UPPAAL, and [109] or [100], both based on EVENT-B, the later being a rather extensive and complete account of the problem. In [77], the authors resort to PVS to generate executable code from the formalisation of an interface provided by a legal document. The popularity of the insulin pump example as a specification case study is also witnessed by the discussion of a Z specification in the recent edition of I. Sommerville's classic on Software Engineering [101].

To the best of our knowledge all those approaches fix, from the outset, a specification logic, entailing a particular perspective on the IIP system. The strategy put forward in this paper has a different starting point:

[☆] Accepted authors' manuscript published as: A. Madeira, R. Neves, M. A. Martins, L, S. Barbosa *A method for rigorous design of reconfigurable systems*, Science of Computer Programming, Volume 132, Pag. 50-76, 2016. [DOI:10.1016/j.scico.2016.05.001]. The final publication is available at Elsevier via `http://www.sciencedirect.com/science/article/pii/S0167642316300272`

(a) Firstly, we recognise that the project of an IIP device involves different classes of requirements which are better expressed through different logic formalisms. For example, requirement *"at all times the insulin flow cannot surpass a given limit"* calls for some sort of quantification and an ordering relation, whereas the statement '*'if the insulin reservoir is almost empty then alarm must sound higher"* suggests the use of a fuzzy language to capture the intended meaning of *almost* and *higher*. On the other hand, some variant of temporal logic becomes necessary to express properties over long-term executions of the device, including typical safety and liveness requirements.

(b) Secondly, we regard an IIP as a prime example of a *reconfigurable* device, *i.e.* one whose execution modes, and not only the values stored in its internal memory, may change in response to the continuous interaction with the environment. Such systems behave differently in different modes of operation, or *configurations*, and commute between them along their lifetime.

Clearly, the dynamics of reconfiguration of a software system, understood here in the sense of statically programmed change of operation mode, can be described by some sort of transition system, whose states represent configurations and transitions are triggered by whatever conditions enforce the move from a configuration to another. Indeed, requirement (b) above entails the need for a design method able to express such a local/global dichotomy, by specifying both the individual configurations and the reconfiguration dynamics. However, one needs also to capture the specific, *local* requirements which characterise each configuration and distinguish one from the others. Formally, such different behaviours can be modelled by imposing additional structure upon states in the transition system which expresses the overall dynamics. Requirement (a), on its turn, suggests such a method to be agnostic *w.r.t.* whatever logic is used to specify the system configurations: the nature of the requirements relevant for a particular view should lead the choice of a suitable base logic.

1.2. The approach

The method proposed in this paper aims at being a step in this direction, by addressing the following research question: *can a rigorous approach to the design of reconfigurable systems be developed based on the hybridisation of the logics used for their local specifications?* The notion of hybridisation and its methodological use are explained below. For the moment, note that the *motto* of the envisaged approach

reconfigurations as transitions, configurations as local models

embodies a two-layered abstraction between a *local* specification stage (that of the system's individual configurations) and a *global* one (concerning the dynamics of reconfiguration). In short, models for reconfigurable software are structured transition systems described within appropriate logical systems. Their states are the individual configurations with whatever structure they have to bear in concrete applications. Transitions correspond to the admissible reconfigurations.

Regarding reconfigurations as *transitions* suggests some sort of modal logic as the language to express them. A modal logic, however, has no ability to explicitly refer to individual configurations. This view justifies the use of *hybrid logic* [64, 17, 22, 6] as the basic language to express evolution (through *modalities*) and locality (through *nominals*). Actually, a specification for this sort of system should be able to make assertions both about the transition dynamics and, locally, about each particular configuration. This leads to the adoption of hybrid logic as the specification *lingua franca* for the envisaged methodology.

In general, hybrid logic adds to a modal language the ability to to name, or to explicitly refer to specific states of the underlying Kripke structure. This is done through the introduction of propositional symbols of a new sort, called *nominals*, each of which is true at exactly one possible state. The sentences are then enriched in two directions. On the one hand, nominals are used as simple sentences holding exclusively in the state they name. On the other hand, explicit reference to states is provided by sentences $@_i \rho$, stating the validity of ρ at the state named i. One may therefore specify (local) properties of specific configurations in the system or even to assert the equality between two particular configurations, something which is beyond what can be said in a modal language. Modalities, however, capture state transitions, providing a way to specify the *global* dynamics of reconfigurability.

Historically, hybrid logic was introduced by A. Prior in his book [92]. However, its seminal ideas emerged by the end of the 50's, in a discussion of C.A. Meredith [17]. The theme was latter revisited, in the school of Sofia, by S.

Passy and T. Tinchev [90]. It achieved global interest within the modal logic community on the 90's, being contributed by P. Blackburn, C. Areces, B. ten Cate, T. Braüner, T. Bolander, among many others (see, e.g., [5, 31, 22, 20]). This lifted the *status* of hybrid logic to an independent branch of modern logic. For an historical account we suggest [17, 22], as well as [18] for a comparison with the original perspective of A. Prior.

On the other hand, identifying configurations, or operation modes, as *local models* emphasises that no special restrictions should be put to their specification. This means that it would not be enough to take a particular version of hybrid logic for writing specifications. Actually, the relevant research question is how to combine a hybrid language with whatever logic seems appropriate to express *local* requirements for each configuration. Specific problems may require specific logics to describe their configurations (*e.g.*, equational, first-order, fuzzy, etc.). Therefore, instead of choosing a particular version of hybrid logic, the method proposed here starts by choosing a specific logic for expressing requirements at the configuration level. This is later taken as the *base* logic on top of which the characteristic features of hybrid logic are developed.

This process is called *hybridisation* [76, 46] and was introduced in the first author's PhD thesis [70]. The basic idea is quite simple: to develop, on top of a given logic, called the *base* logic and framed as an institution [53, 41], the characteristic features of hybrid logic, both at the level of syntax (i.e. modalities, nominals, etc.) and semantics (i.e., possible worlds), together with first-order encodings of such hybridised institutions. In particular, given a *base* institution 'encodable' in the institution of theories in first-order logic, the *hybridisation* method provides a systematic construction of a similar encoding for the corresponding hybridised institution.

To proceed in a completely *generic* way, the whole approach is framed in the context of the theory of institutions of Goguen and Burstall [28, 41], each logic (base and hybridised) treated abstractly as an institution. Institutions provide a precise way to transport specifications and proofs from one logic to another, which allows us to seek for appropriate tool support for simulating and validating our conjectures. The HETS framework offers such a support as discussed in [87] and exemplified later in the paper.

As it will become clearer along the paper, the general idea underlying the design method proposed here is depicted in Figure 1. The upper part refers to the global level of a specification; the lower one to the local description of configurations. The line in the middle emphasises the role of hybrid features: in a formula they provide a way to name evaluation states, whereas in a model they index the relevant configuration. Actually, each configuration is characterised by a local model capturing its functionality and behaviour.

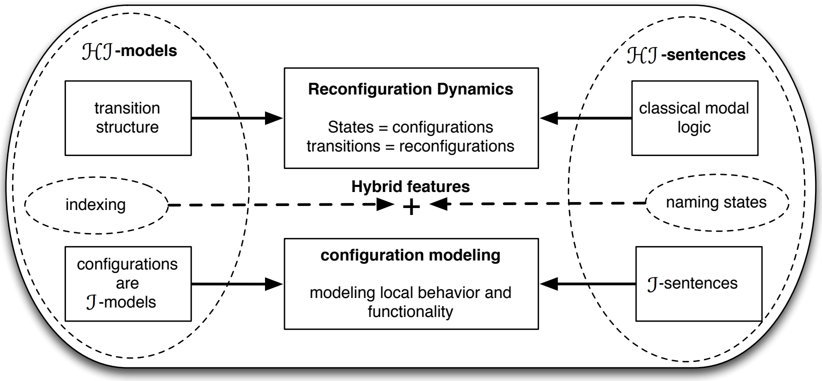


Figure 1: Specification of reconfigurable systems: The approach.

1.3. Contribution and paper's structure

The *hybridisation* process mentioned above, conducted in an institutional setting, is the foundational work on top of which the specification method proposed in this paper was developed. The paper itself is devoted to the detailed introduction of the specification method, illustrated with fragments of the IIP case study. In this sense, it extends the authors' previous work [71, 73]. The former reference is a first introduction to the use of hybrid logics for specification of reconfigurability, whereas the latter discusses an important, but somehow complementary aspect of the design method proposed here — that of requirements elicitation through the use of a suitable set of boilerplates.

In this setting the paper's contributions are twofold:

- The introduction, in a detailed way, of a method, based on hybridisation, for the design of reconfigurable systems (Sections 3 and 4).
- The technical development of a mechanism for capturing a system's *enduring* temporal properties (Section 5).

Sections 3, 4 and 5 contain, as explained above, the paper's original contributions. In all of them fragments of the IIP design are used for illustration purposes. Before that an introduction to the systematic construction of hybrid(ised) logics is provided in Section 2. This includes a brief summary to the institutional rendering of logics. Such a background section seems necessary to set the scene for the paper, exemplifying the generation of several logics used in later sections, and providing relevant pointers to the literature. Finally, related work and perspectives for further research are discussed in Section 6.

2. Background: Hybrid(ised) logics

2.1. Logics as institutions

The development of a systematic process for *hybridising* logics entails the need for a mathematical framework where logics are suitably handled as generic objects. The theory of institutions [53] provides such a framework in a category theoretic setting. The notion of an *institution* formalises that of a logical system, encompassing syntax, semantics and satisfaction. It was put forward by J. Goguen and R. Burstall, in the late seventies, in response to the *"population explosion among the logical systems used in Computing Science"* [53].

The theory proved effective and resilient as witnessed by the wide number of logics formalised in this framework. Examples range from classical logics (propositional, equational, first order, etc.), to the ones underlying specification and programming languages or used for describing particular systems from different domains. Well-known examples include *probabilistic logics* [13], *quantum logics* [29], *hidden and observational logics* [27, 14], *coalgebraic logics* [32], as well as logics for reasoning about *process algebras* [86], *functional* [98, 99] and *imperative programing languages* [98].

Seeking for abstract characterisations of basic conceptual tools, *e.g.*, translations and combination of logics, the theory had a relevant impact in theoretical Computer Science. In particular, it led to the development of a solid *institution-independent specification theory*, on which, structuring and parameterisation mechanisms, required to scale up software specification methods, are defined 'once and for all', irrespective of the concrete logic used in each application domain [103].

Some basic definitions are recalled below and illustrated with a few examples to which we return later in the paper. Most of the technical details presented are not essential for an appreciation of the design method proposed in the paper, and can be skipped on a first reading. The interested reader, however, is referenced to [41] for an extensive account of the theory.

An *institution*

$$\mathcal{I} = (\mathrm{Sign}^{\mathcal{I}}, \mathrm{Sen}^{\mathcal{I}}, \mathrm{Mod}^{\mathcal{I}}, (\models^{\mathcal{I}}_{\Sigma})_{\Sigma\in|\mathrm{Sign}^{\mathcal{I}}|})$$

consists of

- a category $\mathrm{Sign}^{\mathcal{I}}$ whose objects are *signatures* and arrows *signature* morphisms;
- a functor $\mathrm{Sen}^{\mathcal{I}} : \mathrm{Sign}^{\mathcal{I}} \to \mathbb{S}et$ giving for each signature a set of *sentences* over that signature;
- a functor $\mathrm{Mod}^{\mathcal{I}} : (\mathrm{Sign}^{\mathcal{I}})^{op} \to \mathbb{C}AT$, giving for each signature Σ a category whose objects are Σ-*models*, and arrows are Σ-*(model) homomorphisms*; each arrow $\varphi : \Sigma \to \Sigma' \in \mathrm{Sign}^{\mathcal{I}}$, (*i.e.*, $\varphi : \Sigma' \to \Sigma \in (\mathrm{Sign}^{\mathcal{I}})^{op}$) is mapped into a functor $\mathrm{Mod}^{\mathcal{I}}(\varphi) : \mathrm{Mod}^{\mathcal{I}}(\Sigma') \to \mathrm{Mod}^{\mathcal{I}}(\Sigma)$ called a *reduct functor*, whose effect is to cast a model of Σ' as a model of Σ;
- a relation $\models^{\mathcal{I}}_{\Sigma} \subseteq |\mathrm{Mod}^{\mathcal{I}}(\Sigma)| \times \mathrm{Sen}^{\mathcal{I}}(\Sigma)$ for each $\Sigma \in |\mathrm{Sign}^{\mathcal{I}}|$, called the *satisfaction relation*,

such that for each morphism $\varphi : \Sigma \to \Sigma' \in \mathrm{Sign}^{\mathcal{I}}$, the satisfaction condition

$$M' \models^{I}_{\Sigma'} \mathrm{Sen}^{\mathcal{I}}(\varphi)(\rho) \;\text{ iff }\; \mathrm{Mod}^{\mathcal{I}}(\varphi)(M') \models^{I}_{\Sigma} \rho \tag{1}$$

holds for each $M' \in |\mathrm{Mod}^{\mathcal{I}}(\Sigma')|$ and $\rho \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$. Graphically,

$$\begin{tikzcd}
\Sigma \arrow[d, "\varphi"'] & \mathrm{Mod}^{\mathcal{I}}(\Sigma) \arrow[r, dash, "\models^{\mathcal{I}}_{\Sigma}"] & \mathrm{Sen}^{\mathcal{I}}(\Sigma) \arrow[d, "\mathrm{Sen}^{\mathcal{I}}(\varphi)"] \\
\Sigma' & \mathrm{Mod}^{\mathcal{I}}(\Sigma') \arrow[u, "\mathrm{Mod}^{\mathcal{I}}(\varphi)"] \arrow[r, dash, "\models^{\mathcal{I}}_{\Sigma'}"'] & \mathrm{Sen}^{\mathcal{I}}(\Sigma')
\end{tikzcd}$$

A *presentation* in $\mathcal{I}$ consists of a pair (Σ, Γ) where Σ is a $\mathcal{I}$-signature and Γ is a set of Σ-sentences.

The whole theory of institutions was developed to provide a sound way of relating logics and transporting specifications between them. Technically, this is achieved through *comorphisms* which are a special kind of mappings between institutions. Formally, a comorphism between $\mathcal{I}$ an $\mathcal{I}'$ consists of a triple $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ where Φ is a functor between $\mathrm{Sign}^{\mathcal{I}}$ and $\mathrm{Sign}^{\mathcal{I}'}$ and $\alpha : \mathrm{Sen} \Rightarrow \Phi; \mathrm{Sen}'$ and $\beta : \Phi^{\mathrm{op}}; \mathrm{Mod}' \Rightarrow \mathrm{Mod}$, two natural transformations which map sentences and models between both institutions assuring the following *satisfaction condition* – for any signature $\Sigma \in |\mathrm{Sign}|$, for any $\Phi(\Sigma)$-model M' and, for all the Σ-sentences ρ:

$$\beta_\Sigma(M') \models^{\mathcal{I}}_{\Sigma} \rho \;\text{ iff }\; M' \models^{\mathcal{I}'}_{\Phi(\Sigma)} \alpha_\Sigma(\rho)\ .$$

A comorphism is *conservative* if its β component is a surjection. This being the case one has that for any set of sentences $\{\rho\} \cup \Gamma \subseteq \mathrm{Sen}^{I}(\Sigma)$

$$\Gamma \models^{\mathcal{I}}_{\Sigma} \rho \;\text{ iff }\; \alpha_\Sigma(\Gamma) \models^{\mathcal{I}'}_{\Phi(\Sigma)} \alpha_\Sigma(\rho)\ .$$

In practice, the existence of a conservative comorphism connecting two logics allows the transport of proof support mechanisms from one to the other. In the context of the design method proposed Section 3, this is used to allow for the verification of properties expressed in whatever specification logic was found appropriate for a given problem using standard theorem provers for more popular logics.

Another relevant issue for the hybridisation process is the way quantifications are dealt with in institutional terms. In the light of the *lemma on constants* of first order logic, (see e.g. [62]), a set of variables X for a signature can be regarded as a set of constants in its X-expansion. Hence,

$$M \models^{FOL}_{(S,F,P)} (\forall X)\rho \text{ iff } M' \models^{FOL}_{(S,F+X,P)} \rho\ ,$$

for any $(S, F + X, P)$-expansion M' of M. In an institutional setting these expansions can be understood as models M' such that $M = \mathrm{Mod}^{FOL}(i)(M')$ for $i : (S, F, P) \hookrightarrow (S, F + X, P)$ an inclusion morphism. This motivates the introduction of the following quantification principle: given a morphisms $\chi : \Sigma \to \Sigma'$,

$$M \models^{\mathcal{I}}_{\Sigma} (\forall\chi)\rho \text{ iff } M' \models^{\mathcal{I}}_{\Sigma'} \rho\ ,$$

for any $M' \in \mathrm{Mod}^{\mathcal{I}}(\Sigma')$ such that $\mathrm{Mod}^{\mathcal{I}}(\chi)(M') = M$.

In order to assure the functoriality of $\mathrm{Sen}^{\mathcal{I}}$, the quantification morphism needs to be carefully introduced. In the context of first order logic this means that $\mathrm{Sen}^{FOL}(\varphi)(\forall X \rho) = (\forall X^{\varphi})\, \mathrm{Sen}^{FOL}(\varphi')(\rho)$, where X^{φ} and φ'define the pushout:

$$\begin{tikzcd}
(S, F, P) \arrow[r, "\varphi"] \arrow[d, "X"'] & (S, F', P') \arrow[d, "X^{\varphi}"] \\
(S, F \uplus X, P) \arrow[r, "\varphi'"'] & (S, F' \uplus X^{\varphi}, P')
\end{tikzcd}$$

where $X^{\varphi} = \{x : \varphi_{st}(s) | x : s \in X\}$, and φ' is the canonical expansion of φ mapping each $x : s$ in $x : \varphi_{st}(s)$.

2.2. *The hybridisation process*

As mentioned in the Introduction, the hybridisation process [76, 46, 70] which underlies the design method proposed in this paper, is based on the representation of logics as institutions. It enriches a base (arbitrary) institution $\mathcal{I}$ with hybrid logic features and the corresponding Kripke semantics. The result is still an institution, $\mathcal{HI}$ — the *hybridisation of* $\mathcal{I}$. This subsection reviews a simplified version of this process, *i.e.*, quantifier-free and non-constrained, in order to convey the basic intuitions.

At the syntactic level the base signatures are enriched with nominals and polyadic modalities. Therefore, the category of $\mathcal{I}$*-hybrid signatures*, denoted by $\mathrm{Sign}^{\mathcal{HI}}$, is defined as the direct (cartesian) product of categories of the original category of signatures $\mathrm{Sign}^{\mathcal{I}}$ and that of signatures of *REL*, the sub-institution of (the institution of) first order logic, without non-constant operation symbols, Sign^{REL}. Signatures of the hybridised institution combine those of $\mathcal{I}$ with a family of constants Nom for *nominals* and a set of relational symbols Λ to represent *modalities*. $\mathcal{HI}$ signatures are, thus, triples $(\Sigma, \mathrm{Nom}, \Lambda)$, with signature morphisms $\varphi = (\varphi_{\mathrm{Sign}}, \varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}}) : \ (\Sigma, \mathrm{Nom}, \Lambda) \to (\Sigma', \mathrm{Nom}', \Lambda')$, defined component-wise: inherited from $\mathcal{I}$ for the first component, given as maps relating nominals and modalities, and preserving the arities of the latter as expected.

The second step in the method is to enrich the base sentences accordingly. The sentences of the base institution $\mathcal{I}$ and the nominals in Nom are taken as atoms and composed with the Boolean connectives, the modalities given in Λ, and satisfaction operators indexed by nominals. For example, for a n-ary modality λ, a nominal i and $\rho, \rho_1, \rho_2 \ldots, \rho_n$ $\mathcal{HI}$ sentences, the following are also sentences in $\mathcal{HI}$: $[\lambda](\rho_1, \ldots, \rho_n), \langle\lambda\rangle(\rho_1, \ldots, \rho_n)$ and $@_i\rho$.

Given a $\mathcal{HI}$-signature morphism φ, the translation of sentences $\mathrm{Sen}^{\mathcal{HI}}(\varphi)$ is defined structurally: e.g., $\mathrm{Sen}^{\mathcal{HI}}(\varphi)(i) = \varphi_{\mathrm{Nom}}(i)$, $\mathrm{Sen}^{\mathcal{HI}}(\varphi)(@_i\rho) = @_{\varphi_{\mathrm{Nom}}(i)}\mathrm{Sen}^{\mathcal{HI}}(\rho)$ and $\mathrm{Sen}^{\mathcal{HI}}(\varphi)([\lambda](\rho_1, \ldots, \rho_n)) = [\varphi_{\mathrm{MS}}(\lambda)](\mathrm{Sen}^{\mathcal{HI}}(\rho_1), \ldots, \mathrm{Sen}^{\mathcal{HI}}(\rho_n))$.

Turning to semantics, models of $\mathcal{HI}$ can be regarded as (Λ-)Kripke structures whose worlds are $\mathcal{I}$-models. Formally, they are pairs (M, W) where W is a (Nom, Λ)-model in *REL* and M is a function which assigns to each state $w \in |W|$ a model in $M_w \in |\mathrm{Mod}^{\mathcal{I}}(\Sigma)|$.

In each model (M, W), W_n provides interpretation for nominal n, whereas relation W_λ interprets modality λ. The reduct definition is lifted from the base institution: the reduct of a Δ'-model (M', W') along with a signature morphism $\varphi : \Delta \to \Delta'$ is the Δ-model (M, W) such that W is the $(\varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}})$-reduct of W' (i.e, $|W| = |W'|$, $W_n = W'_{\varphi_{\mathrm{Nom}}(n)}$, for each nominal n, and $W_\lambda = W'_{\varphi_{\mathrm{MS}}(\lambda)}$ for each modality in Λ) and for each $w \in |W|$, $M_w = \mathrm{Mod}^{\mathcal{I}}(\varphi_{\mathrm{Sign}})(M'_w)$.

Finally, the satisfaction relation for the hybridised institution resorts to the one in the base institution for sentences in $\mathcal{I}$, *i.e.*,

- $(M, W) \models^w \rho$ iff $M_w \models^{\mathcal{I}} \rho \quad$ when $\rho \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$,

captures the semantics of nominals

- $(M, W) \models^w i$ iff $W_i = w \;$, when $i \in \mathrm{Nom}$,
- $(M, W) \models^w @_i\rho$ iff $(M, W) \models^{W_i} \rho$,

and modalities, as in

- $(M, W) \models^w [\lambda](\xi_1, \ldots, \xi_n)$ iff, for any $(w, w_1, \ldots, w_n) \in W_\lambda$, $(M, W) \models^{w_i} \xi_i$ for some $1 \leq i \leq n$,

and is defined as usual for the Boolean connectives. Note that, whenever the base institution has Boolean connectives, there is a replication of these connectives in its hybridisation: those in the base institution sentences (that we denote by $\circledwedge$, $\circledvee$, $\circleddarrow$ and $\circlearrowleft$) and those introduced by the hybridisation method ($\wedge$, $\vee$, $\Rightarrow$ and $\neg$). However, as shown in [46, 70], they semantically collapse, in the sense that for any $\rho, \rho' \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$ we have that $(M, W) \models^w \neg\rho$ iff $(M, W) \models^w \circleddash\rho$ and that $(M, W) \models^w \rho \star \rho'$ iff $(M, W) \models^w \rho \circledast \rho'$ for any $\star \in \{\wedge, \vee, \Rightarrow\}$. In virtue of this fact, we will not make any denotational distinction between these two levels of connectives

The main result is that $\mathcal{HI}$ effectively constitutes an institution [76]. The next step is the systematic characterisation of encodings of $\mathcal{HI}$ into the institution of many sorted first-order logic (*FOL*) building on existent encodings of into *FOL* of the base institution $\mathcal{I}$.

Quantified Hybridisation. Extending this procedure to quantified sentences requires the introduction of an adequate quantification space $\mathcal{D}^{\mathcal{HI}}$ for $\mathrm{Mod}^{\mathcal{HI}}$ such that, its projection in the first component belongs in a quantification space $\mathcal{D}^{\mathcal{I}}$ adequate for $\mathrm{Mod}^{\mathcal{I}}$. This is regarded as an additional parameter in the hybridisation process.

As a subclass of $\mathrm{Sign}^{\mathcal{HI}}$, the quantification morphisms consists of triples $\chi = (\chi_{\mathrm{Sign}}, \chi_{\mathrm{Nom}}, \chi_{\mathrm{MS}}) : (\Sigma, \mathrm{Nom}, \Lambda) \to (\Sigma', \mathrm{Nom}', \Lambda')$. Each of these components is responsible for a particular kind of quantification. For example, considering $\chi_{\mathrm{Nom}} : \mathrm{Nom} \hookrightarrow \mathrm{Nom} + Y$, for Y a finite set of constants, and considering χ_{MS} the identity morphism, we obtain the standard state quantification of the literature.

The hybridisation process is adjusted to the quantified version as follows: the set $\mathrm{Sen}^{\mathcal{HI}}(\Delta)$ is enriched, for any $\chi : \Delta \to \Delta' \in \mathcal{D}^{\mathcal{HI}}$ and $\rho \in \mathrm{Sen}^{\mathcal{HI}}(\Delta')$, with the sentence $(\forall\chi)\rho$; the translation of sentences is extended, in each morphism $\varphi : \Delta \to \Delta_1$, by $\mathrm{Sen}^{\mathcal{HI}}(\varphi)((\forall\chi)\rho) = (\forall\chi(\varphi))\mathrm{Sen}^{\mathcal{HI}}(\varphi[\chi])(\rho)$; finally, for the satisfaction, we consider

- $(M, W) \models^{w} (\forall\chi)\rho$ iff $(M', W') \models^{w} \rho$ for any (M', W') such that $\mathrm{Mod}^{\mathcal{HI}}(\chi)(M', W') = (M, W)$.

Existential quantification is introduced in a similar way.

Observe that, in these cases, it is necessary to distinguish between quantification coming from the base institution and the one introduced by the hybridisation process, itself. The relation between both levels is given by

- $(M, W) \models^{w} (\text{\textcircled{$\forall$}}\chi)\rho$ implies that $(M, W) \models^{w} (\forall(\chi, 1_{\mathrm{Nom}}, 1_{\Lambda}))\rho$, and
- $(M, W) \models^{w} (\exists(\chi, 1_{\mathrm{Nom}}, 1_{\Lambda}))\rho$ implies that $(M, W) \models^{w} (\text{\textcircled{$\exists$}}\chi)\rho$.

Constrained Models. Introduced in [44], represent a third parameter in the hybridisation process, providing a precise way to put constraints upon models of the hybridised logics. Typical constraints come from sharing among local universes or the rigidification of the variables. They are particularly important to ensure the existence of proper encodings.

Formally, a *constrained models functor for* $\mathcal{HI}$ consists of a subfunctor $\mathrm{Mod}^{C} \subseteq \mathrm{Mod}^{\mathcal{HI}}$ (*i.e.*, such that $\mathrm{Mod}^{C}(\Delta)$ is a subcategory of $\mathrm{Mod}^{\mathcal{HI}}(\Delta)$, for any $\Delta \in |\mathrm{Sign}^{\mathcal{HI}}|$) that reflects weak amalgamation, *i.e.*, such that any pushout in $\mathrm{Sign}^{\mathcal{HI}}$ which forms an amalgamation square for $\mathrm{Mod}^{\mathcal{HI}}$, is also a weak amalgamation square for Mod^{C}. The models of $\mathrm{Mod}^{C}(\Delta)$ are called *constrained model of* $\mathcal{HI}$. Replacing $\mathrm{Mod}^{\mathcal{HI}}$ by Mod^{C} in $\mathcal{HI}$ we obtain an institution, denoted by $\mathcal{HI}^{C}$ [44].

2.3. Hybridisation examples

Example 2.1 (The trivial institution). *The simplest institution one can think of is TRIV. Its category of signatures,* Sign^{TRIV}*, is the* final category, i.e.*, the category whose class of objects is the singleton set* $\{*\}$ *and morphisms reduce to the identity* $1_*(*) = *$*. The sentences functor,* Sen^{TRIV}*, sends object* $*$ *into the empty set* $\emptyset$ *and, morphism* 1_**, into the empty function. The models functor,* Mod^{TRIV}*, sends the signature* $*$ *to the final category. Since the set of sentences is empty, the satisfaction condition holds trivially.*

Let us consider the free-hybridisation of TRIV. The signature category corresponds to

$$\mathrm{Sign}^{TRIV} \times \mathrm{Sign}^{REL} \;\cong\; \mathrm{Sign}^{REL} .$$

Since $\mathrm{Sen}^{TRIV}(*) = \emptyset$*,* $\mathrm{Sen}^{\mathcal{H}TRIV}(*, \mathrm{Nom}, \Lambda)$ *is the set of sentences built up from nominals in* Nom *by the application of modalities in* Λ *and Boolean connectives. This kind of formulas are called* pure hybrid formulas *in [19, 64]. Models of* $\mathrm{Mod}^{\mathcal{H}TRIV}(*, \mathrm{Nom}, \Lambda)$ *are relational structures* (W, M)*, where* M *is the constant function* $M_w = *$*, for any* $w \in |W|$*.*

It is an usual assumption to consider initial states in the models. For this case, we may consider the 'pointed' version of $\mathcal{H}TRIV$*, denoted by* $\mathcal{H}TRIV\mathcal{P}$ *as the institution obtained considering by taking* $\mathrm{Sign}^{\mathcal{H}TRIV\mathcal{P}} = \mathrm{Sign}^{\mathcal{H}TRIV\mathcal{P}}$*,* $\mathrm{Sen}^{\mathcal{H}TRIV\mathcal{P}} = \mathrm{Sen}^{\mathcal{H}TRIV\mathcal{P}}$ *and assuming for each* $\Delta \in |\mathrm{Sign}^{\mathcal{H}TRIV\mathcal{P}}|$*, the category whose objects* $|\mathrm{Mod}^{\mathcal{H}TRIV\mathcal{P}}(\Delta)| = \{((M, W), s) | (M, W) \in \mathrm{Mod}^{\mathcal{H}TRIV}$ *and* $s \in |W|\}$ *and whose morphisms are the morphisms of* $\mathrm{Mod}^{\mathcal{H}TRIV}(\Delta)$ *that maps initial states to initial states. Then,*

$$((M, W), s) \models^{\mathcal{H}TRIV\mathcal{P}} \rho \textit{ iff } (M, W)(\models^{\mathcal{H}TRIV})^{s}\rho .$$

An interesting institution for the specification of hierarchical state transition systems is obtained through the hybridisation of $\mathcal{H}TRIV\mathcal{P}$*, which we denote (simply) by* $\mathcal{H}^{2}TRIV$*. Models of this institution are "Kripke structures* *of Kripke structures". Thus $\mathcal{H}^2$TRIV signatures are triples* $((*, \mathrm{Nom}^l, \Lambda^l), \mathrm{Nom}^u, \Lambda^u)$ *with* Nom^l,Λ^l *and* Nom^u, Λ^u *denoting the nominals and the modalities of the lower and upper layers of hybridisation, respectively. Hence, the set of sentences* $\mathrm{Sen}^{\mathcal{H}^2 TRIV}((*, \mathrm{Nom}^l, \Lambda^l), \mathrm{Nom}^u, \Lambda^u)$ *is defined by the grammar*

$$\Phi \ni \phi \mid I \mid @_I \Phi \mid \langle M \rangle (\Phi, \ldots, \Phi) \mid [M] (\Phi, \ldots, \Phi) \mid \neg\Phi \mid \Phi \odot \Phi\,, \tag{2}$$

where

$$\phi \ni i \mid @_i \phi \mid \langle m \rangle (\phi, \ldots, \phi) \mid [m] (\phi, \ldots, \phi) \mid \neg\phi \mid \phi \odot \phi\,, \tag{3}$$

for $I \in \mathrm{Nom}^u$, $M \in \Lambda^u_n$, $i \in \mathrm{Nom}^l$, $m \in \Lambda^l_n$ *and* $\odot \in \{\wedge, \vee, \to\}$. *In order to prevent ambiguities, we use upper and lower case letters to distinguish the nominals and modalities symbols of these two levels. Since they collapse, we do not need to make denotational distinction in the two levels of Boolean connectives. Our tagging convention is extended also to $\mathcal{H}^2$TRIV models: a* $((*, \mathrm{Nom}^l, \Lambda^l), \mathrm{Nom}^u, \Lambda^u)$*-model is denoted by* (M^u, W^u) *where, for any* $w \in |W^u|$, *the models* M^u_w *are denoted by* $((W^l_w, M^l_w), s_w)$.

Example 2.2 (*EQ*, *FOL*, *PL* **and** *PA*). *Signatures in the institution EQ of equational logic are pairs* (S, F) *where S is a set of sort symbols and* $F = \{F_{\underline{\mathrm{ar}} \to s} \mid \underline{\mathrm{ar}} \in S^*, s \in S\}$ *is a family of sets of operation symbols indexed by arities* $\underline{\mathrm{ar}}$ *(for the arguments) and sorts s (for the results).* Signature morphisms *map both components in a compatible way: they consist of pairs* $\varphi = (\varphi^{\mathrm{st}}, \varphi^{\mathrm{op}}) : \ (S, F) \to (S', F')$, *where* $\varphi^{\mathrm{st}} : \ S \to S'$ *is a function, and* $\varphi^{\mathrm{op}} = \{\varphi^{\mathrm{op}}_{\underline{\mathrm{ar}} \to s} : \ F_{\underline{\mathrm{ar}} \to s} \to F'_{\varphi^{\mathrm{st}}(\underline{\mathrm{ar}}) \to \varphi^{\mathrm{st}}(s)} \mid \underline{\mathrm{ar}} \in S^*, s \in S\}$ *is a family of functions mapping operations symbols respecting arities.*

A model M for a signature (S, F) *is an algebra interpreting each sort symbol s as a carrier set* M_s *and each operation symbol* $\sigma \in F_{\underline{\mathrm{ar}}} \to s$ *as a function* $M_\sigma : M_{\underline{\mathrm{ar}}} \to M_s$, *where* $M_{\underline{\mathrm{ar}}}$ *is the product of the arguments' carriers. Model morphisms are homomorphisms of algebras,* i.e., *S-indexed families of functions* $\{h_s : \ M_s \to M'_s \mid s \in S\}$ *such that for any* $m \in M_{\underline{\mathrm{ar}}}$, *and for each* $\sigma \in F_{\underline{\mathrm{ar}} \to s}$, $h_s(M_\sigma(m)) = M'_\sigma(h_{\underline{\mathrm{ar}}}(m))$. *For each signature morphism* φ, *the* reduct *of a model* M', *say* $M = \mathrm{Mod}^{EQ}(\varphi)(M')$ *is defined by* $(M)_x = M'_{\varphi(x)}$ *for each sort and function symbol x from the domain signature of* φ. *The models functor maps signatures to categories of algebras and signature morphisms to the respective reduct functors.*

Sentences are universally quantified equations $(\forall X)t = t'$. *Sentence translations along a signature morphism* $\varphi : (S, F) \to (S', F')$, i.e., $\mathrm{Sen}^{EQ}(\varphi) : \mathrm{Sen}^{EQ}(S, F) \to \mathrm{Sen}^{EQ}(S', F')$, *replace symbols of* (S, F) *by the respective* φ*-images in* (S', F'). *The sentences functor maps each signature to the set of its sentences and each signature morphism to the respective sentences translation.*

The satisfaction relation is the usual Tarskian satisfaction defined recursively on the structure of the sentences as follows:

- $M \models_{(S,F)} t = t'$ *when* $M_t = M_{t'}$, *where* M_t *denotes the interpretation of the* (S, F)*-term t in M defined recursively by* $M_{\sigma(t_1, \ldots, t_n)} = M_\sigma(M_{t_1}, \ldots, M_{t_n})$.
- $M \models_{(S,F)} (\forall X)\rho$ *when* $M' \models_{(S,F+X)} \rho$ *for any* $(S, F + X)$*-expansion* M' *of M.*

Signatures of $\mathcal{H}$*EQ are triples* $((S, F), \mathrm{Nom}, \Lambda)$ *and the sentences are defined as in (2) but taking* (S, F)*-equations* $(\forall X)t = t'$ *as atomic base sentences. Models are Kripke structures with a (local)-*(S, F)*-algebra per world.*

The institution of (multi-sorted) first-order logics, denoted by FOL, is defined as EQ but considering also an S^**-family of symbols P interpreted as predicates in the models. The models are defined as in EQ but interpreting any* $\pi \in P_{\underline{\mathrm{ar}}}$ *as a predicate* $M_\pi \subseteq M_{\underline{\mathrm{ar}}}$. *The sentence of FOL are defined as usual, taking beyond the equations, predicate formulas, its compositions through Boolean connectives and quantified sentences. The satisfaction extends the satisfaction of EQ by taking*

- $M \models^{FOL}_{S,F,P} \pi(t_1, \ldots, t_n)$ *iff* $(M_{t_1}, \ldots, M_{t_n}) \in M_\pi$ *and*
- $M \models^{FOL}_{(S,F,P)} (\forall X)\rho$ *if* $M' \models^{FOL}_{(S,F+X,P)} \rho$ *for any* $(S, F + X, P)$*-expansion* M' *of M (see discussion in the above section).*

The hybridisation of FOL constrained to the the models with sharing in the universes shared and variables realisation was suggested in [71] as suitable logic for the reconfigurable systems specification. Observe that the institution of propositional logic *PL can be formalised as the fragment of FOL obtained by restricting the signatures with empty sets of sorts (see [41]). The fragment of the quantified free hybridisation of PL with an unique binary modality,* i.e., Λ_1 *is a singleton and* $\Lambda_n = \emptyset$ *for* $n \neq 1$*, corresponds to the standard propositional hybrid logic of the literature. Another useful hybridisation in this 'family' is the hybridisation of the institution PA of partial algebras. Models of this institution are defined as above, but considering partial, instead of total, functions. Versions of hybridisations FOL and PA with sharing and rigidification of components are carefully treated in [70] and are denoted in the sequel by* $\mathcal{H}$*FOLR and* $\mathcal{H}$*PAR.*

Example 2.3 (*MVL*$_L$)**.** *Multi-valued logics [55] generalise classic logics by replacing, as its* truth domain*, the 2-element Boolean algebra by larger sets structured as* complete residuate lattices*. They were originally formalised as institutions in [3] (see also [43] for a recent reference).*

Residuate lattices *are tuples* $L = (\boldsymbol{L}, \leq, \wedge, \vee, \top, \bot, \otimes)$*, where*

- $(\boldsymbol{L}, \wedge, \vee, \top, \bot)$ *is a lattice ordered by* $\leq$*, with carrier* $\boldsymbol{L}$*, with (binary) infimum* ($\wedge$) *and supremum (* $\vee$)*, and biggest and smallest elements* $\top$ *and* $\bot$*;*
- $\otimes$ *is an associative binary operation such that, for any elements* $x, y, z \in L$*,*
  - $x \otimes \top = \top \otimes x = x$ *,*
  - $y \leq z$ *implies that* $(x \otimes y) \leq (x \otimes z)$ *,*
  - *there exists an element* $x \Rightarrow z$ *such that* $y \leq (x \Rightarrow z)$ *iff* $x \otimes y \leq z$ *.*

  The residuate lattice L is complete if any subset $S \subseteq \boldsymbol{L}$ *has infimum and supremum denoted by* $\bigwedge S$ *and* $\bigvee S$*, respectively.*

Given a complete residuate lattice L, the institution MVL$_L$ *is defined as follows.*

- *MVL*$_L$*-signature are PL-signatures.*
- *Sentences of MVL*$_L$ *consist of pairs* (ρ, p) *where p is an element of L and* ρ *is defined as a PL-sentence over the set of connectives* $\{\Rightarrow \vee, \top, \bot, \otimes\}$*.*
- *A MVL*$_L$*-model M is a function* $M : FProp \to L$ *.*
- *For any* $M \in \mathrm{Mod}^{MVL_L}(FProp)$ *and for any* $(\rho, p) \in \mathrm{Sen}^{MVL_L}(FProp)$ *the satisfaction relation is*

  $$M \models^{MVL_L}_{FProp} (\rho, p) \ \textit{ iff } \ p \leq (M \models \rho)\,,$$

  where $M \models \rho$ *is inductively defined as follows:*
  - *for any proposition* $p \in FProp$*,* $(M \models p) = M(p))$ *,*
  - $(M \models \top) = \top$ *,*
  - $(M \models \bot) = \bot$ *,*
  - $(M \models \rho_1 \star \rho_2) = (M \models \rho_1) \star (M \models \rho_2)$*, for* $\star \in \{\vee, \Rightarrow, \otimes\}$ *.*

This institution captures many multi-valued logics in the literature. For instance, taking L as the Łukasiewicz arithmetic lattice over the closed interval $[0, 1]$*, where* $x \otimes y = 1 - max\{0, x + y - 1)\}$ *(and* $x \Rightarrow y = min\{1, 1 - x + y\}$*), yields the standard* propositional fuzzy logic.

The institution obtained through the hybridisation of MVL$_L$*, for a fixed L, is similar to the* $\mathcal{H}$*PL institution defined above, but for two aspects,*

- *sentences are defined as in (2) but considering MVL FProp-sentences* (ρ_0, p) *as atomic;*
- *to each world* $w \in |W|$ *is associated a function assigning to each proposition its value in L.*

It is interesting to note that expressivity increases along hybridisation, even if one restricts to the case of a (one-world) standard semantics. For instance, differently from the base case where each sentence is tagged by an L-value, one may now deal with more structured expressions involving several L-values, as in, for example, $(\rho, p) \wedge (\rho', p')$.

Example 2.4 ($\mathcal{P}PL$). *The probabilised propositional logic ($\mathcal{P}PL$), obtained from the application of the probabilisation method [9], assigns a probability value to each PL–sentence. Formally,*

- *$\mathcal{P}PL$–signatures are PL–signatures,*
- *$\mathcal{P}PL$–sentences are defined by the grammar,*

  $$\rho \ \ni \ t < t \,|\, \neg\rho \,|\, \rho \Rightarrow \rho\,,$$

  where

  $$t \ \ni \ r \,|\, \int \beta \,|\, t + t \,|\, t\,.\,t\,,$$

  for $r \in \mathbb{R}$ and β a PL–sentence.
- *and $\mathcal{P}PL$–models are tuples (S, F, P, V) where (S, F, P) is a probability space and V a function from the possible outcomes (elements of S) to PL–models.*

Then, the probability value of a given PL–sentence ($\int \beta$) is equal to the sum of all the probabilities assigned to the outcomes that, through function V, point to a PL–model satisfying β.

Through hybridisation $\mathcal{P}PL$ gives rise to the $\mathcal{H}\mathcal{P}PL$ logic where,

- *$\mathcal{H}\mathcal{P}PL$–signatures are triples $(Prop, Nom, \Lambda)$,*
- *$\mathcal{H}\mathcal{P}PL$–sentences are generated by the grammar*

  $$\rho \ \ni \ \rho_0 \,|\, i \,|\, @_i\rho \,|\, \rho \odot \rho \,|\, \neg\rho \,|\, \langle\lambda\rangle(\rho, \dots, \rho) \,|\, [\lambda](\rho, \dots, \rho)\,,$$

  where ρ_0 is a $\mathcal{P}PL$–sentence, $\odot \in \{\wedge, \vee, \rightarrow\}$, i a nominal and λ a modality,
- *and $\mathcal{H}\mathcal{P}PL$–models are Kripke structures with a (local) $\mathcal{P}PL$–model per world.*

Note that along this process two levels of Boolean connectives are added, which, as expected, semantically collapse. It is interesting to notice, however, that the atomic connectives cannot be collapsed into the new ones because the latter only act outside the scope of the $\int$ operator, while the former only apply within.

2.4. First-order encodings

The first steps in defining a method for generating first-order encodings for hybridised institutions were presented in [76] and further extended to presentations, constrained and quantified models in [46, 70]. In particular, for any institution 'encodable' in presentations in *FOL*, the method constructs an encoding from its hybridisation to *FOL*. Therefore, a wide variety of computer assisted provers for *FOL* can be 'borrowed' to reason about specifications in the new, hybridised logics. Technically such encodings are achieved by extending the classical *standard translation* of modal logic into the (one-sorted) first order logic [104], more precisely, for its hybrid version [17], to the encodings of hybridised institutions into *FOL*.

The general idea of the standard translation from $\mathcal{H}PL$ into the (one-sorted) first-order logic, is to consider a sort to denote the state space, where nominals are interpreted as constants, modalities as binary relations, and propositions as unary predicates (where $p(w)$ means that the proposition p holds at state w). The idea underlying the standard translation $\mathcal{H}FOL2FOL$ (e.g. [22]), is to extend this encoding by considering a new universe ST as an extra sort in the signature, and 'flattening' the universes, operations and predicates of the (local) *FOL*-models into a unique (global) *FOL*-model. Local functions and predicates become parametric over states, and the state universes distinguished with a sort-family of definability predicates. Intuitively, whenever m belongs to the universe of w, $\pi(w, m)$ and $\sigma(w, m) = b$ means that $\pi(m)$ and $\sigma(m) = b$ hold in the state w. Moreover, restricting this global model M to the local universes, operations and predicates of a fixed world w, we obtain a "slice of M", say $M|_w$, which consists of a local *FOL*-model representing (and coinciding with) M_w.

The general method is based on the application of a state-parametrisation construction of $\mathcal{H}FOL2FOL$ to lift $\mathcal{I}2FOL$ to $\mathcal{HI}2FOL$, where $\mathcal{I}$ in these acronyms stands for an arbitrary institution — that of the base logic to be hybridised. Thus, signatures and sentences targeted by $\mathcal{I}2FOL$ become parametric on states and the remaining sentences are treated as in $\mathcal{H}FOL2FOL$. A slice $M|_w$ corresponds now to the "*FOL*-interpretation" of the local $\mathcal{I}$-model M_w, which can be recovered using $\mathcal{I}2FOL$.

3. The method

The hybridisation process discussed in section 2, provides an extensive number of hybrid(ised) logics, several of them with appropriate first-order encodings, which are the basis of a specification method for reconfigurable systems. This section introduces such a method, offering a sort of guided tour to its application through the IIP example.

The method, summarised in Figure 2, is divided into four steps: *i)* definition of the specification *framework*, *ii) interface* description, *iii)* specification of *properties* and *iv)* analysis and *validation*. The next four subsections provide a 'guided tour' through these states illustrating them with the specification of a IIP device.

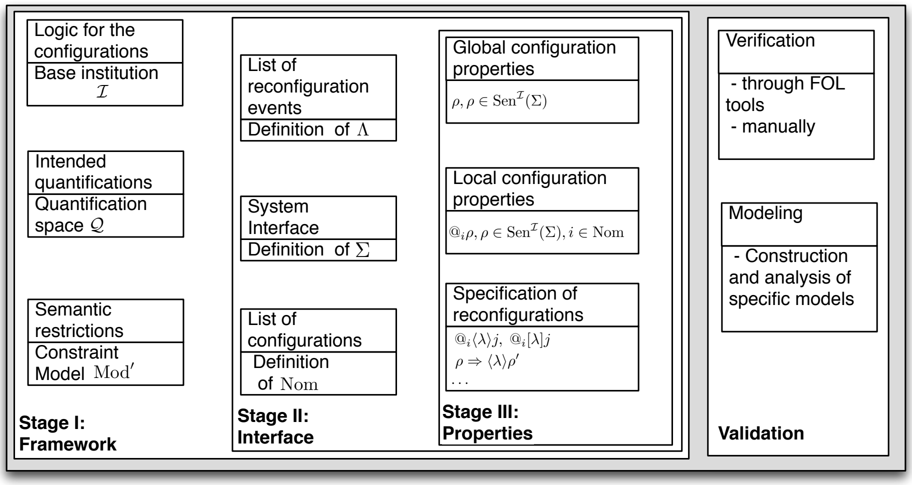

Logic for the configurations
Base institution $\mathcal{I}$
Intended quantifications
Quantification space $\mathcal{Q}$
Semantic restrictions
Constraint Model Mod'
Stage I: Framework
List of reconfiguration events
Definition of Λ
System Interface
Definition of Σ
List of configurations
Definition of Nom
Stage II: Interface
Global configuration properties
$\rho, \rho \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$
Local configuration properties
$@_i\rho, \rho \in \mathrm{Sen}^{\mathcal{I}}(\Sigma), i \in \mathrm{Nom}$
Specification of reconfigurations
$@_i\langle\lambda\rangle j,\ @_i[\lambda]j$
$\rho \Rightarrow \langle\lambda\rangle\rho'$
$\ldots$
Stage III: Properties
Verification
- through FOL tools
- manually
Modeling
- Construction and analysis of specific models
Validation

Figure 2: Specification of reconfigurable systems: The method.

3.1. *Definition of the specification framework*

In this first stage the Software Engineer defines the specification framework by generating the specification language for the local configurations through the hybridisation of a base logic. This entails the need for choosing

- the institution representing the *base logic* found suitable to specify individual configurations of the problem at hand,
- the *quantification space*, to express which kinds of quantifications are necessary, and
- the relevant *constrained models*, to capture additional semantic constraints.

These choices are crucial as they fix the working institution and therefore constrains all subsequent development. In particular, the choice of the base institution $\mathcal{I}$ needs to be made only after most of the problem informal requirements were given and clearly understood. The approach proposed in this paper is largely independent of the specific logic chosen to specify concrete configurations. One may, for example, specify them as *multialgebras* to cope with non-determinism, or with *multi-valued* logic to deal with uncertainty. Another possibility is using *partial equational* logic to deal with exceptions, or *observational* logics when the local level encapsulates hidden spaces. Moreover, as mentioned above, in some cases each configuration can be regarded as a transition system itself and a modal language is in then order. Actually, different logics have been suggested for software specification (see e.g. [16]) and, from the point of view of our method, there is no reason to favour one over the other. The only possible limitation is related to tool support: for example, choosing a base logic for which a translation to FOL or CASL exists, may be a good option if tool support is na issue, as such a translation can be lifted to the resulting hybridised logic.

The two other choices are equally relevant. The decision about what kind of quantification is to be allowed has a direct impact on the expressiveness of the framework. On the other hand, enforcing additional constraints upon the models provides the technical support to deal with sharing (e.g. of data, operations or both) across configurations. Such constraints may also tune the accessibility relation which expresses the reconfiguration dynamics (imposing, for example, reflexivity on an ordered structure). Both issues are closely related: for example, to obtain suitable encodings, global quantification over the universes of individual configurations requires the previous choice of a suitable constrained model. In practice the following kinds of quantification are found useful:

- *Quantification over the base institution*, i.e. over the domains of system's configurations. This can be
  - *global*, if a shared universe for all the configurations is chosen and the quantified variables are assumed to be rigid. Actually, this is the typical way to 'relate computations of' or to 'communicate values across' different configurations.
  - or *local*, i.e., ranging over each particular configuration domain. Technically, this is obtained by taking a quantifier-free hybridisation *w.r.t.* the first component of the quantification morphisms.
- *Quantification over nominals*, which makes it possible to express properties about the systems global state space. This is particularly useful, for instance, to express the existence of configurations satisfying a given requirement.
- *Quantification over modalities*, a rather powerful form of quantification, which, in general, allows us to control in a very precise way the underlying accessibility relations.

As usual, when facing these choices the specifier should take into account the compromise between expressiveness and formal tractability. For instance, quantification over modalities should not be used if the aim is to take advantage of the first-order encodings discussed in [46, 70] to obtain suitable tool support for validating the specifications. Similarly, the relationships between the choice of constrained models and that of quantification spaces cannot be overlooked.

Example: *The* IIP *device*

In the IIP device case study, we identify in the original requirements [111, 112] two types of insulin infusion:

- *basal*, where insulin is continuously injected throughout the day at adjustable rates;
- and *bolus*, which is a one–time insulin delivery rapidly administered, closely resembling an injection.

The former is used as the default, with insulin administration rates given by a 24h-profile that takes into account the scheduling of the patient meals, sleep and exercise. When unsuitable, such profile may be temporarily replaced. The latter adds up to the basal mode and is typically used to cover up unexpected situations or high-carb food intake. We also consider an additional suspension state where any kind of insulin administration is forbidden. Transitions among these different states are triggered by specific events. Their structure, however, depends on which is the current profile of insulin administration chosen for the patient. Each such profile corresponds to an upper-level state which constitutes a configuration of the IIP device.

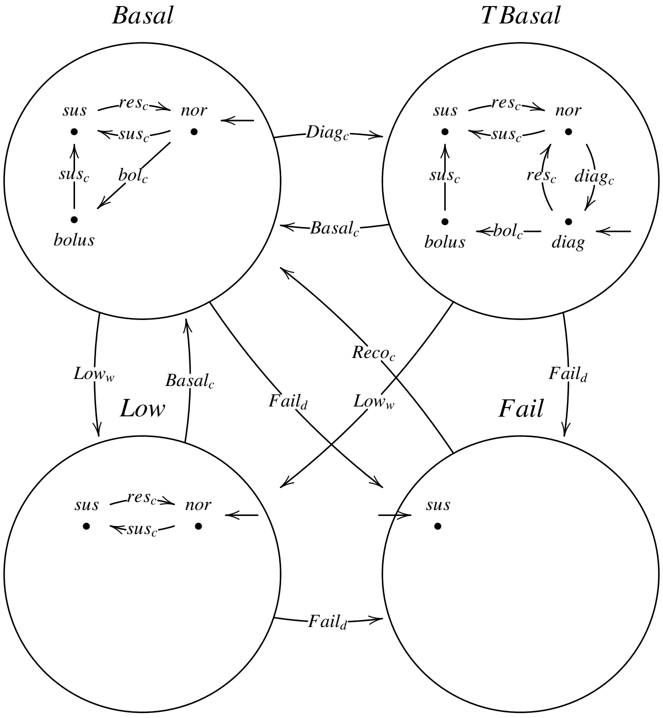


Figure 3: Hierarchical state transition system for the IIP device.

The overall structure of the IIP specification is that of an *hierarchical transition system* as depicted in Figure 3. Actually, each configuration is given as a transition system; switching between them provides a higher level transition structure. If $\mathcal{H}TRIV$ is appropriate to specify plain transition systems, as the ones populating each local configuration, its hybridisation $\mathcal{H}^2TRIV$ will express the global dynamics and the change from one profile to another.

3.2. Interface description

At this second stage all the relevant 'vocabulary' to specify the intended system is declared. This includes the enumeration of all configurations to be considered (i.e., component Nom) and the events triggering reconfigurations (i.e., component Λ). Moreover, a (local) $\mathcal{I}$-signature for the specification of individual configurations (i.e., component Σ) is also required. In a sense, the latter can be understood as the actual interface of the system since all services and functions offered at each configuration must be declared at this stage.

Hybrid signatures are represented in the sequel in a notation similar to that of the specification language CASL [8], according to the following structure:

spec SPECNAME **in H**BASEINST =
  Nom
    declaration of nominal symbols
  Modal
    declaration of modality symbols
  BaseSig
    signature of the base institution
  Axioms
    set of axioms

SPECNAME is the specification identifier, and BASEINST stands for the base institution $\mathcal{I}$ adopted when fixing the specification framework. Entries **Nom** and **Modal** are used for the declaration of nominal and modality symbols, respectively. Arities of modalities are given by natural numbers placed in front of the respective symbol (e.g. *event:k* means that modality *event* has arity *k*). The parameter **BaseSig** is the declaration of the $\mathcal{I}$-signature Σ. Since the structure of those local signatures is specific for each hybridisation, there is no way to fix the syntax for this declaration. Finally, the field **Axioms** contains the specification expressed in axioms of the chosen logic.

Example: ***The* IIP *device***

Returning to our running example, we first identify the four upper states corresponding to different insulin administration profiles:

- $Basal$, denoting the pump following a 24-h basal profile.
- $TBasal$, representing the temporary basal profile that overrides basal due to specific circumstances. Extra care is expected in this profile: a new diagnosis is required before injecting insulin.
- *Low*, which is triggered when the pump detects that insulin reservoir levels are becoming low, and therefore any form of extra insulin doses (as is the case of bolus) are forbidden.
- $Fail$, triggered whenever faulty behaviour is encountered and thus entails the prohibition of any sort of insulin administration.

Switching between these administration profiles is accomplished by the following transitions which encode reconfiguration commands: $Basal_c$, to evolve to the *Basal* profile, $Diag_c$ to change to a diagnosis requiring profile, $Fail_d$ and $Reco_c$ to signal the device failure and recovery, respectively. The warning associated to a low insulin level is denoted by Low_w.

Inside each upper-state, the configuration behaviour is given again by a transition system whose states correspond to different infusion modes, namely,

- *nor*, denotes insulin infusion with rates being given by a specific profile;
- *bolus*, adds up to *nor* through bolus injections;
- *sus*, simply denotes suspension which entails insulin infusion to not occur;
- *diag*, represents a diagnosis process, which while occurring also forbids any kind of insulin infusion.

All of these components are collected in the following hybrid signature:

spec IIPSPEC **in H**HTRIV =
  Nom
    Basal, *TBasal*, *Low*, *Fail*
  Modal
    $Diag_c$: **1**, $Basal_c$: **1**, Low_w: **1**, $Fail_d$: **1**, $Reco_c$: **1**
  BaseSign {
    Nom
    sus, *nor*, *bolus*, *diag*
    Modal
    res_c: **1**, sus_c: **1**, bol_c: **1**, $diag_c$: **1**
    }

$\mathcal{HI}$-boilerplate	$\mathcal{HI}$-representation
System has modes <set of *Nom*>	$\bigvee_{n \in Nom} n$
If P then property Q holds	$P \Rightarrow Q$
If P then property Q does not hold	$P \Rightarrow \neg Q$
<*Mode*> is active	$Mode$
System changes from $< Mode1 >$ to $< Mode2 >$ through $< Event >$	$@_{Mode1}\langle Event\rangle Mode2$
All transitions through $< Event >$ lead to $< Mode >$	$[Event]Mode$
Property P holds in $< Mode >$	$@_{Mode}P$
There are no transitions through $< Event >$	$\neg\langle Event\rangle\top$
...	...

Figure 4: Typical boilerplates for the specification of reconfigurable systems.

3.3. Specification of properties

All system properties, both at the local and global levels, are introduced at this stage. They correspond to requirements placed at different levels of abstraction. In particular, one has to consider

1. the global properties, i.e., properties holding in all the configurations, which are expressed through (atomic) $\mathcal{I}$-sentences;
2. the reconfiguration dynamics whose specification resorts to the modal features introduced through the hybridisation process;
3. and finally, the local properties, i.e. those relative to specific configurations, which are expressed by tagging $\mathcal{I}$-sentences with the satisfaction operator @ ($@_i\rho$ is used to express that property ρ holds in the configuration named by i).

An alternative way of expressing these properties is by resorting to a suitable pallet of *boilerplates* which translate to sentences to the relevant logic. This provides a 'user friendly' syntax to express properties rigorously. The design of boilerplates for the specification of reconfigurable systems, some of them shown in Figure 4, and their formal semantics is discussed in [73].

Example: *The* IIP *device*

The first properties to be considered in the specification of the IIP, are the ones that rule out 'anomalous' models. Actually, as it always happens in loose semantics, a specification may admit other models beyond the ones in which all configurations are distinguished and suitably identified by nominals. Such *'junk configurations'* exist, but one may also have to deal with *'confusion on the configurations'* whenever different nominals identify the same state. Expressing no junk through boilerplates leads to

System has modes $Basal, TBasal, Low, Fail$,

which translates into the sentence

$$Basal \vee TBasal \vee Low \vee Fail\ .$$

Similarly, the statement imposing *no confusion* between profiles *Basal* and *TBasal* is given by boilerplate

If $Basal$ is active then property $\underline{TBasal \text{ is active}}$ does not hold.

which corresponds to

$$\neg @_{Basal}\ TBasal\ .$$

This generalises to

$$\bigwedge_{i,j\in \mathrm{Nom}, i\neq j} \neg @_i j \ ,$$

to extend to every possible pair of nominals.

Once global properties are established we turn to the specification of the reconfigurations dynamics. In the IIP example this amounts to the definition of the upper level transitions. Those are the transitions which encode the reconfiguration dynamics, i.e. the change of an execution mode to another one. As explained above, this is an example of a system whose configurations can themselves be modelled as transition structures (the lower level ones). Adding the reconfiguration layer leads to a hierarchy of such structures, as illustrated in Figure 3. Through boilerplates one may express a $Diag_c$ transition between $Basal$ and $TBasal$:

System changes from $Basal$ to $TBasal$ through event $Diag_c$.

or that any $Diag_c$ transition from $Basal$ will *necessarily* lead to $TBasal$:

Property $\underline{\text{all transitions through } Diag_c \text{ lead to } TBasal}$ holds in $Basal$.

Formally, the former is expressed by

$$@_{Basal}\langle Diag_c\rangle TBasal$$

and the restriction that from $Basal$ one may only reach $TBasal$ by means of $Diag_c$ is written as

$$@_{Basal}[Diag_c]TBasal \ .$$

Both properties can be "compressed" into a conjunction

$$@_{Basal}(\langle Diag_c\rangle TBasal \wedge [Diag_c]TBasal) \ ,$$

which we abbreviate, through notation $\langle M\rangle^{\dagger}\rho \equiv (\langle M\rangle\rho \wedge [M]\rho)$, into,

$$@_{Basal}\langle Diag_c\rangle^{\dagger}\ TBasal \ .$$

Finally, the statement cutting all $Fail_d$ transitions in $Fail$ is simply formulated as,

$$@_{Fail}\neg\langle Fail_d\rangle\top \ ,$$

where $\top$ stands for a taulology in the language, for instance $@_{Fail}\,Fail$.

Finally, let us consider the local properties of configurations. As in this example they are themselves described in $\mathcal{H}TRIV$, each configuration corresponds to the specification of a Kripke structure. Each of them is endowed with an initial state to define the local (lower level) state which is expected to become active when a reconfiguration forces switching to the corresponding profile. For instance, on a transition to the $Basal$ configuration, state nor will become active. The following conjunction in $\mathcal{H}^2TRIV$ establishes the entry point in each of the four profiles considered:

$$@_{Fail}sus \ \wedge \ @_{Basal}nor \ \wedge \ @_{Low}nor \ \wedge \ @_{TBasal}diag \ .$$

Transitions must now also be defined *w.r.t.* each of the available profiles. For instance, in $Basal$, there exists a sus_c transition between nor and sus,

$$\text{Property } \underline{\text{system changes from } \textit{nor} \text{ to } \textit{sus} \text{ through event } \textit{sus}_c} \text{ holds in } \textit{Basal}$$

and there are no sus_c transitions departing from *sus*,

$$\text{P} = \text{Property } \underline{\text{there are no transitions through } \textit{sus}_c} \text{ holds in sus.}$$
$$\text{Property } \underline{\text{P}} \text{ holds in Basal}$$

The direct use of $\mathcal{H}^2TRIV$ sentences provides an alternative to boilerplates, whenever the specifier prefers to avoid such a structured text format. From a methodological point of view, however, the rationale is similar. This means that properties concerning the lower level are made *w.r.t.* each of the available profiles. For example, the 'no confusion' condition in *Basal* between *nor* and *sus* is formulated as,

$$@_{Basal}\ @_{nor}\neg sus\ ,$$

and, as before, it may be generalised to hold between *nor* and any other state,

$$@_{Basal}\ @_{nor}(\neg sus \wedge \neg bolus)$$

The definition of local transitions is relative to the upper state to which they belong. For example, that a sus_c transition exists from *nor* to *sus* in the *Basal* profile is given by

$$@_{Basal}\ @_{nor}\langle sus_c\rangle\ sus\ ,$$

Similarly, that no sus_c transition from *sus* is possible at *Basal* is defined as

$$@_{Basal}\neg @_{sus}\langle sus_c\rangle\ \top\ .$$

Note that cutting transitions may become an increasingly cumbersome task, as the number of modalities increases. For example to specify that no further transitions are possible once inside the *Fail* profile, one writes

$$@_{Fail}\ \neg(\langle sus_c\rangle\top \vee \langle diag_c\rangle\top \vee \langle bol_c\rangle\top \vee \langle res_c\rangle\top)\ .$$

The complete $\mathcal{H}^2TRIV$-specification is given in Figure 5.

3.4. Analysis and validation

The construction and the analysis of particular models of a specification is a fundamental step in the design process. In a sense, it can be understood as a high level implementation of the specification, a first prototype acting as a proof-of-concept for the system. Once a model is available, its validation, i.e. the systematic verification of the specified properties, becomes crucial. Although this can be done with 'paper and pencil', the availability of computational proof-support tools is a necessary condition for the methodology to be considered a viable alternative in the software industry.

There are a number of provers available for propositional hybrid logics (which can, of course, be of use when dealing with hybrid(ised) propositional logic), for example, HTab [63], HyLoTab [105] and Spartacus [56]). Other works, for example [69, 61], study model checking procedures for hybrid propositional models.

Unfortunately, propositional hybrid logic has a limited use in the specification of reconfigurable systems. Actually, it only suits the case in which states have a very simple structure, i.e. when the local description of individual configurations is irrelevant. On the other hand, and to the best of our knowledge, there is no dedicated tool support for richer hybrid logics.

spec IIPSPEC **in** **H**HTRIV =

Nom

$Basal$, $TBasal$, Low, $Fail$

Modal

$Diag_c$: **1**, $Basal_c$: **1**, Low_w: **1**, $Fail_d$: **1**, $Reco_c$: **1**

BaseSign {

Nom

sus, nor, $bolus$, $diag$

Modal

res_c: **1**, sus_c: **1**, bol_c: **1**, $diag_c$: **1**

}

Axioms

- $Basal \vee TBasal \vee Low \vee Fail$ **%**(no junk)**%**
- $Basal \Rightarrow (\neg\, TBasal \wedge \neg\, Low \wedge \neg\, Fail)$ **%**(no confusion wrt Basal)**%**
- $TBasal \Rightarrow (\neg\, Low \wedge \neg\, Fail)$ **%**(no confusion wrt TBasal)**%**
- $Low \Rightarrow (\neg Fail)$ **%**(no confusion wrt Low)**%**

- $(@_{Basal}\, \langle Diag_c\rangle$” $TBasal\,) \wedge (\langle Diag_c\rangle\top \Rightarrow\ Basal\,)$
- $(@_{Fail}\, \langle Reco_c\rangle^\dagger\ Basal\,) \wedge (\langle Reco_c\rangle\top \Rightarrow\ Fail\,)$
- $(TBasal \vee Low\,) \Rightarrow \langle Basal_c\rangle^\dagger\ Basal$
- $\langle Basal_c\rangle\top \Rightarrow (TBasal \vee Low)$
- $(\neg Fail \Rightarrow \langle Fail_d\rangle^\dagger Fail\,) \wedge (\langle Fail_d\rangle\top \Rightarrow \neg Fail\,)$
- $(Basal \vee TBasal \Rightarrow \langle Low_w\rangle^\dagger\ Low\,) \wedge (\langle Low_w\rangle\top \Rightarrow Basal \vee TBasal\,)$

- $@_{Fail}\ sus \wedge @_{Basal}\ nor \wedge @_{Low}\ nor \wedge @_{TBasal}\ diag$ **%**(the initial states)**%**

- $@_{Fail}\ \neg\, (\langle sus_c\rangle\top \vee \langle diag_c\rangle\top \vee \langle bol_c\rangle\top \vee \langle res_c\rangle\top\,)$

- $@_{Basal}\, ((@_{nor}\langle sus_c\rangle^\dagger\ sus\,) \wedge (@_{bolus}\langle sus_c\rangle^\dagger\ sus\,) \wedge (\neg\, @_{sus}\langle sus_c\rangle\top\,)\,)$
- $@_{Basal}\, ((@_{sus}\langle res_c\rangle^\dagger\ nor\,) \wedge (\neg\, @_{nor}\langle res_c\rangle\ \top\,) \wedge (\neg\, @_{sus}\langle res_c\rangle\top\,)\,)$
- $@_{Basal}\, ((@_{nor}\langle bol_c\rangle^\dagger\ bolus\,) \wedge (\neg\, @_{bolus}\langle bol_c\rangle\ \top\,) \wedge (\neg\, @_{sus}\langle bol_c\rangle\top\,)\,)$
- $@_{Basal}\, ((\neg\, @_{nor}\langle diag_c\rangle\ \top\,) \wedge (\neg\, @_{bolus}\langle diag_c\rangle\ \top\,) \wedge (\neg\, @_{sus}\langle diag_c\rangle\top\,)\,)$

- $@_{Low}\, ((@_{nor}\langle sus_c\rangle^\dagger\ sus\,) \wedge (\neg\, @_{sus}\langle sus_c\rangle\ \top\,)\,)$
- $@_{Low}\, ((@_{sus}\langle res_c\rangle^\dagger\ nor\,) \wedge (\neg\, @_{nor}\langle res_c\rangle\ \top\,)\,)$
- $@_{Low}\, ((\neg\, @_{nor}(\langle diag_c\rangle\top \vee \langle bol_c\rangle\top\,)\,) \wedge (\neg\, @_{sus}(\langle diag_c\rangle\top \vee \langle bol_c\rangle\top\,)\,)\,)$

- $@_{TBasal}\, ((@_{nor}\langle sus_c\rangle^\dagger\ sus\,) \wedge (@_{bolus}\langle sus_c\rangle^\dagger\ sus\,) \wedge (\neg\, @_{sus}\langle sus_c\rangle\top\,) \wedge (\neg\, @_{diag}\langle sus_c\rangle\top\,)\,)$
- $@_{TBasal}\, ((@_{sus}\langle res_c\rangle^\dagger\ nor\,) \wedge (@_{diag}\langle res_c\rangle^\dagger\ nor\,) \wedge (\neg\, @_{nor}\langle res_c\rangle\top\,) \wedge (\neg\, @_{bolus}\langle res_c\rangle\top\,)\,)$
- $@_{TBasal}\, ((@_{diag}\langle bol_c\rangle^\dagger\ nor\,) \wedge (\neg\, @_{nor}\langle bol_c\rangle\ \top\,) \wedge (\neg\, @_{sus}\langle bol_c\rangle\top\,) \wedge (\neg\, @_{bolus}\langle res_c\rangle\top\,)\,)$
- $@_{TBasal}\, ((@_{nor}\langle diag_c\rangle^\dagger\ diag\,) \wedge (\neg\, @_{diag}\langle diag_c\rangle\ \top\,) \wedge (\neg\, @_{sus}\langle diag_c\rangle\top\,) \wedge (\neg\, @_{bolus}\langle diag_c\rangle\top\,)\,)$

end

Figure 5: The IIP specification.

The method proposed in this paper takes a different path. In order to prototype a specification written in an hybrid(ised) logic, or to validate its consistency, the latter is translated into *first-order* logic (*FOL*), so that the software engineer can take advantage of several provers already available for *FOL*. Actually, the institution-based framework underlying the hybridisation process described in Section 2, which provides a whole pallet of (hybrid) logics for translating the system requirements, also offers for free the conceptual machinery for this translation to *FOL*, whenever it exists. Then, the prover toolset HETS [85], a framework specifically designed to support specifications expressed in different institutions, offers suitable proof support. Using a common metaphor [84], HETS may be seen as a "motherboard" where different "expansion cards" can be plugged. These pieces are individual logics (with their particular analysers and proof tools) as well as logic translations, suitably encoded in the theory of institutions. HETS already integrates parsers, static analysers and provers for a wide set of individual logics and manages heterogeneous proofs resorting to the so-called graphs of logics, i.e., graphs whose nodes are logics and, whose edges, are comorphisms between them.

The existence of a suitable translation, technically a *comorphism*, from an hybrid(ised) logic $\mathcal{HI}$ to *FOL*, gives, for free, access to a number of provers integrated in HETS. Those include, for example, SPASS [107], VAMPIRE [94] and DARWIN [12]. Such a translation, as noticed above, is not available for all logics. However, in [76, 46], the authors provide a roadmap for addressing this issue: In [76], the authors show that the hybridisation of an institution with a comorphism to *FOL* also has a comorphism to *FOL*. Then, in reference [46], the authors extend this result and characterise conservativity of those translations to define in which cases it is possible to borrow, in an effective way, proof support from *FOL*. Note that the proof of this result is constructive, offering a method to implement such a translation. In practice, this is a very general, broadly applicable result since several specification logics have a comorphism to *FOL*. Such is the case, for example, of propositional, equational, first-order, modal or even hybrid logic, among many others.

Several other features of HETS can be explored in the context of the methodology proposed here. For instance, the model finder of DARWIN may be used as a consistency checker for specifications. On the other hand, recent encodings of *FOL* into HASCASL, a specification language for functional programs, open new perspectives for prototyping generated specifications in a standard programming language such as HASKELL.

Example: *The* IIP *device*

Properties of the IIP device, at different levels of abstraction, can be verified through a suitable translation to *FOL*, using, for example, the SPASS theorem prover. A first example is provided by the property stating that any failure transition followed by a recovery will always lead to basal mode, *i.e.*,

$$[Fail_d][Reco_c]Basal\,,$$

which is translated by HETS into,

$$(\forall w)(\forall v)\; Fail_d(w,v) \Rightarrow (\forall u)\, Reco_c(v,u) \Rightarrow u = Basal\,.$$

Another example is the requirement that the pump is either in failure state or can always fail. In $\mathcal{H}^2TRIV$ this is written as

$$Fail \vee \langle Fail_d\rangle Fail\,,$$

and translated to *FOL* as

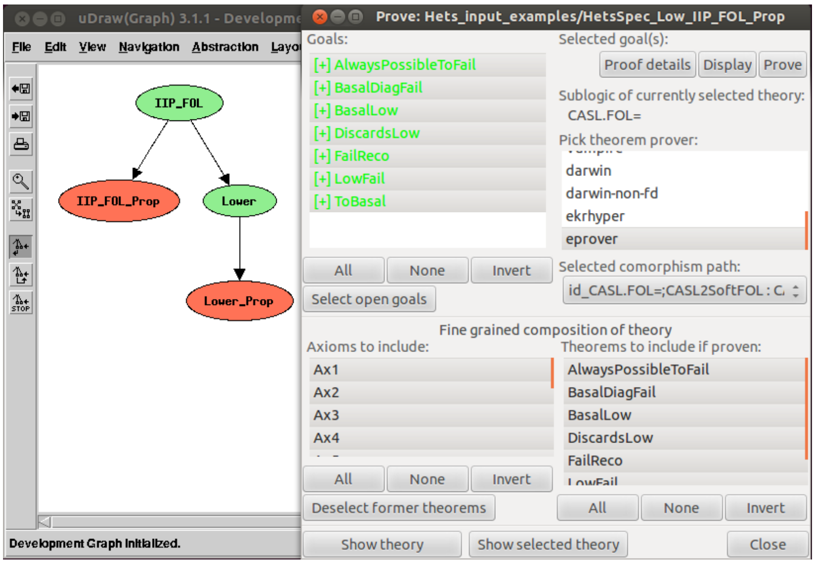
$$(\forall w)\; w = Fail \vee ((\exists v)\; Fail_d(w,v) \wedge v = Fail)\,.$$

Finally, the property that specifies that the pump is either suspended or able to suspend

$$sus \vee \langle sus_c\rangle sus\,,$$

is written in *FOL* as,

$$(\forall w)\; init(w) = sus \vee ((\exists v)\; sus_c(w, init(w), v) \wedge v = sus)\,.$$



Development Graph initialized.

Prove: Hets_input_examples/HetsSpec_Low_IIP_FOL_Prop

Goals:

[+] AlwaysPossibleToFail
[+] BasalDiagFail
[+] BasalLow
[+] DiscardsLow
[+] FailReco
[+] LowFail
[+] ToBasal

Selected goal(s):

Sublogic of currently selected theory:
CASL.FOL=

Pick theorem prover:

darwin
darwin-non-fd
ekrhyper
eprover

Selected comorphism path:
id_CASL.FOL=;CASL2SoftFOL : C.

Fine grained composition of theory

Axioms to include:

Ax1
Ax2
Ax3
Ax4

Theorems to include if proven:

AlwaysPossibleToFail
BasalDiagFail
BasalLow
DiscardsLow
FailReco
LowFail

Figure 6: HETS session.

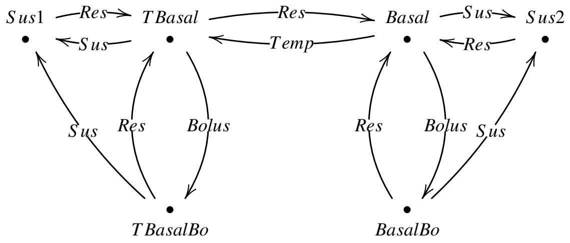


Figure 7: The IIP transition structure.

Figure 6 shows the HETS session where those properties were proved.

4. Different needs, different logics

In Section 3 the specification of a IIP device was presented in $\mathcal{H}^2$*TRIV* characterising a two-level hierarchical transition system. Our aim was to illustrate the step-by-step development of the specification method proposed in this paper. In this section, however, the focus is the choice of specific hybrid(ised) logics to suitably address specific requirements. The message is mirrored in the Section title: to meet a (specification) need, a particular logic can be chosen, hybridised and used. Therefore, configurations will no longer be regarded as transition systems themselves, as before, but as models for the different logics considered.

To guide our exposition we assume that the reconfiguration dynamics of the IIP device considered in the sequel is represented by the transition structure depicted in Figure 7. In comparison with the $\mathcal{H}^2$*TRIV* models in Section 3, note that the behaviour remains similar: We still have the *Basal* and *Temporary basal* profiles, both allowing bolus administration, but now we disregard any sort of diagnosis; suspension is still possible, but is represented through two nominals at the same level. The need for two nominals comes from the insufficient expressivity power *w.r.t.* "remembering" previous states, which in this particular case means that after suspending, unless each relevant state has its own suspension mode, it is impossible to know which was the previous mode.

4.1. Capturing equational and first-order requirements: the insulin flow

As already mentioned, the purpose of the IIP is, as autonomously as possible, to administrate to a patient the adequate insulin quantities. This means that in an IIP specification the precise characterisation of the insulin quantities that the pump effectively debits becomes a major concern. In the sequel, we address this concern first with a simpler logic, *EQ*, and then, due to the nature of the requirements, with a more powerful logic, *FOL*, at the cost of decidability.

One of the safety measures *w.r.t.* insulin flow in a IIP is the definition of an upper limit for the amount of insulin that is injected per time slice. We define this limit as a constant over the naturals,

$$maxFlow : Nat$$

and then change its value along the possible configurations. For instance, when the pump is suspended no insulin should flow, which fixes the maximum flow to zero, *i.e.*,

$$(Sus1 \vee Sus2) \Rightarrow maxFlow = 0 \; .$$

Note that, without loss of generality, the natural numbers are used to represent insulin rates. The conformance of such rates with user programmable *basal* profiles, which inform the insulin quantities to be injected for each instant of time, is highly desirable. We define such programs as a function between time and naturals,

$$basal : Time \rightarrow Nat$$

with time also abstracted as a natural number. Function

$$curFlow : Time \rightarrow Nat$$

gives for each instant the intended quantity of insulin the pump is expected to administer. Finally, stating that the pump should follow the basal profile at specific modes becomes possible. For instance,

$$@_{Basal}(\textcircled{\forall}\, t : Time).\, curFlow(t) = basal(t)$$

imposes that in the *Basal* mode the insulin output should exactly correspond to the basal profile. When a bolus is given, however,

$$@_{BasalBo}(\textcircled{\forall}\, t : Time).\, curFlow(t) = basal(t) + bolus(t) \; ,$$

with *bolus* being defined as a function $bolus : Time \rightarrow Nat$. Two additional remarks on the definition of a profile are in order:

- The IIP is reconfigurable, but profiles are not. That is, user program profiles not taking into account the different internal modes of the pump and it is expected from the profiles to remain the same along all possible configurations. Hybridised logics allows for dealing with this kind of rigidifications. For example in the case of *basal*, using global quantification over the base institution and over nominals (i.e., quantification over states), we can impose that

$$(\forall w, w') \,.\, (\forall t, n : Nat) \,.\, @_w basal(t) = n \Rightarrow @_{w'} basal(t) = n \; .$$

- Having defined profiles as functions imposes on them to be always defined along all configurations. This is clearly an over restriction since there are modes where such profiles are not relevant, as is the case of the suspension mode. One solution for this issue is the switch to a more suitable logic where partial functions are allowed; for instance PAR. Within this logic we may say, for example, that

$$(Sus_1 \vee Sus_2) \Rightarrow (\forall t : Nat).\, \neg df(basal(t)) \; ,$$

  where df is the definability predicate in *PAR*. Naturally in *PAR* the notion of rigidity ought to be made partial: functions become expected not to change their values along the configurations in which they are defined.

Although decidable, $\mathcal{H}$EQ has enough expressivity to formalise a number of interesting requirements. However, there are others whose formalisation entails more powerful logics, even if decidability is lost. A prime example of this is on the requirement that insulin flow cannot surpass specific values. To capture this requires some sort of ordering, which comes naturally if we switch to $\mathcal{H}$FOL, the hybrid(ised) first-order logic, where relations are treated as first–class citizens. Then, the formalisation becomes

$$(\textcircled{$\forall$}\, t : Nat).\, curFlow(t) \leq maxFlow\ .$$

With HETS and the specification language $\mathcal{H}$CASL, of which $\mathcal{H}FOL$ is a sub–logic, several non-trivial properties can be automatically proved. One example is the statement that when the pump is suspended the insulin flow is not bigger than in any other modes:

$$(\forall t : Nat).\ (\forall n : Nat)\ .\ ((@_{Sus1} curFlow(t) = n) \Rightarrow curFlow(t) \geq n)\ .$$

Another example is the requirement that since *bolus* adds up to the *basal* insulin rates, the flow in *Basal* mode cannot surpass the flow in the *BasalBo* mode:

$$(\forall t : Nat)\ .\ (\exists n, n' : Nat)\ .\ @_{Basal}\ curFlow(t) = n \wedge @_{BasalBo}\ curFlow(t) = n' \wedge n \leq n'\ .$$

4.2. Capturing fuzzy and probabilistic reasoning: alarms and air-in-line sensors

Suppose the IIP device has a sensor able to inform if the insulin reservoir is empty or not, represented here, in a first attempt, by a predicate *empty*. In several cases, however, this binary information is not that useful: some action may depend on the reservoir being just with "enough" insulin. Similarly, one might consider an alarm that sounds *louder* when the situations gets "worse", as opposed to the alarm being on or off and the situation being "good" or "bad".

Many–valued logics, *i.e.* logics that can take other truth spaces than the standard Boolean one, are suitable for the situations just described where interesting cases appear in a "grey" area. For the sake of illustration, we consider here the simplest of such logics — the *three-valued* one. The reader is referred to Section 2 for a detailed description of the hybridisation of this sort of logics.

This adds a third element ($^1/_2$) to the Boolean truth space, with $0 \leq {}^1/_2 \leq 1$. Thus, predicate *empty* does not necessarily needs to output one or zero, but can also give one half, which we interpret as the case where the reservoir has an unknown insulin level inside. In the alarm case, we may also represent it in the form of a predicate (named *alarm*), with output zero being interpreted as *turned off*, one half as *on but not sounding loud*, and one as *on and sounding loud*.

With both predicates we become able to address, in a suitable way, situations that motivate a switch to hybrid(ised) many–valued logics. For instance, the requirement that an unknown reservoir state preceded by a full reservoir, activates the alarm with low sound is expressed as

$$@_{Basal}(\forall t : Time)\ .\ \neg(empty(t), {}^1/_2) \wedge (empty(t+1), {}^1/_2) \Rightarrow (alarm(t+2), {}^1/_2)\ .$$

When a bolus is being given, however, the insulin flow, as we have seen before, is typically higher, so, since the situation is probably worse, it might be a good idea to fully activate the alarm, i.e.,

$$@_{BasalBo}(\forall t : Time)\ .\ \neg(empty(t), {}^1/_2) \wedge (empty(t+1), {}^1/_2) \Rightarrow (alarm(t+2), 1)\ .$$

Naturally, when the pump is suspended alarms are not raised, *i.e.*,

$$(Sus_1 \vee Sus_2) \Rightarrow (\forall t : Time)\ .\ \neg(empty(t), {}^1/_2) \wedge (empty(t+1), {}^1/_2) \Rightarrow (alarm(t+2), 0)\ .$$

There are yet other components in a IIP device which motivate the use of different logics. Our last example concerns the detection of air within lines, a situation that may be fatal to the patient if significant quantities of air are injected along with insulin, but that can be avoided through an air-in-line sensor and an air filter. Suppose the sensor reports the probabilities of the air-in-line situation causing problems to the patient. This entails the need for bringing to scene a *probabilistic* logic; the propositional variant is enough for illustration purposes. The translation of probabilistic properties, requiring a combination of FOL with the real numbers, follows the work of P. Baltazar [9].

One of the most important sort of requirements on the air–in–line sensor concerns the definition of the specific conditions which activate the filter, given, as mentioned before, in probabilistic terms. A clear example of this is a requirement stating that in *Basal* mode the filter activates whenever the probability given by the sensor is bigger than x:

$$@_{Basal} \int bubbles > x \Rightarrow \int filter = 1 \; .$$

Naturally, it the pump is suspended there is no risk of air being injected into the patient and therefore the filter does not need to be activated for any value returned by the sensor, *i.e.*,

$$(Sus_1 \vee Sus_2) \Rightarrow \int filter = 0 \; .$$

Typically, due to sundry constraints, specifications are not proved to be always correct but rather to work in an acceptable percentage of cases. This shift from “perfection” to “just good enough”, which is currently a reality in industry, can also be applied here with the help of probabilistic propositional logic. For example, we may state that in the *Basal* mode the probability of the patient getting problems due to air in line is less than x:

$$@_{Basal} \textstyle\int bubbles \wedge \neg filter < x \; .$$

5. Long term properties

Back to our running example, suppose one wants to express the (liveness) requirement that *from any mode, suspension is reachable through a path composed of sus_c transitions*. This is an example of a statement expressed over an entire sequence of transitions and, therefore, are not directly expressible through the modalities considered so far. Those, however, can easily be extended to deal with regular expressions over binary modalities, providing a way to express long term, “enduring” properties. The requirement above could then be written as

$$\langle sus_c^* \rangle \, sus \; .$$

In the sequel we show how the hybridisation $\mathcal{HI}$ of a logic $\mathcal{I}$ can be extended in this direction. This is a common procedure, namely to extend process logics into a temporal dimension [78]. We prove, however, that the result of this enrichment is still an institution $\mathcal{HI}^*$, as defined below.

The signatures category is inherited from $\mathcal{HI}$, i.e., $\mathrm{Sign}^{\mathcal{HI}} = \mathrm{Sign}^{\mathcal{HI}^*}$. The sentences functor $\mathrm{Sen}^{\mathcal{HI}^*}$ is defined for each signature $(\Sigma, \mathrm{Nom}, \Lambda) \in |\mathrm{Sign}^{\mathcal{HI}^*}|$, as the smallest set such that

- $\mathrm{Sen}^{\mathcal{HI}}(\Sigma, \mathrm{Nom}, \Lambda) \subseteq \mathrm{Sen}^{\mathcal{HI}^*}(\Sigma, \mathrm{Nom}, \Lambda)$,
- for any $\rho \in \mathrm{Sen}^{\mathcal{HI}}(\Sigma, \mathrm{Nom}, \Lambda)$ and $a \in Ac(\Lambda)$, where $Ac(\Lambda) ::= \lambda \mid a;a \mid a \cup a \mid a^*$, $\lambda \in \Lambda_1$, we have that $[a]\rho, \langle a \rangle \rho \in \mathrm{Sen}^{\mathcal{HI}^*}(\Sigma, \mathrm{Nom}, \Lambda)$.

Recall Λ_1 is the set of binary modalities in the signature.

For any morphism $\varphi : (\Sigma, \mathrm{Nom}, \Lambda) \to (\Sigma', \mathrm{Nom}', \Lambda') \in \mathrm{Sign}^{\mathcal{HI}^*}$, we define

$$\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(\langle a \rangle \rho) = \langle \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a) \rangle \mathrm{Sen}^{\mathcal{HI}}(\varphi)(\rho)) \; ,$$

where $\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a)$ extends the translation of modalities of $\mathrm{Sen}^{\mathcal{HI}}(\varphi)$ in a structural way:

- $\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(\lambda) = \mathrm{Sen}^{\mathcal{HI}}(\varphi)(\lambda)$, $\lambda \in \Lambda_1$,
- $\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_1; a_2) = \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_1); \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_2)$,
- $\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_1 \cup a_2) = \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_1) \cup \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_2)$,

- $\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a^*) = (\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a))^*$.

Analogously, $\mathrm{Mod}^{\mathcal{HI}^*} = \mathrm{Mod}^{\mathcal{HI}}$ where the interpretation of the regular expressions in a model $(M, W) \in |\mathrm{Mod}^{\mathcal{HI}^*}(\Sigma, \mathrm{Nom}, \Lambda)|$ extends the interpretation of modalities by

- $W_{a_1;a_2} = W_{a_1}; W_{a_2}$,
- $W_{a_1 \cup a_2} = W_{a_1} \cup W_{a_2}$ and
- $W_{a^*} = (W_a)^*$.

where ; and $*$ stand for relational composition and transitive closure upon binary relations. Then the satisfaction $\models^{\mathcal{HI}^*}$ extends $\models^{\mathcal{HI}}$ with

- $(M, W) \models^w \langle a \rangle \rho$ iff $(M, W) \models^{w'} \rho$ for some $(w, w') \in W_a$ and
- $(M, W) \models^w [a]\rho$ iff $(M, W) \models^{w'} \rho$ for any $(w, w') \in W_a$.

Finally, we observe that the satisfaction condition holds in $\mathcal{HI}^*$, as stated below.

Theorem 5.1. *Let $\mathcal{HI}^*$ be the dynamic extension of the hybridisation of $\mathcal{I}$, $\varphi : (\Sigma, \mathrm{Nom}, \Lambda) \to (\Sigma', \mathrm{Nom}', \Lambda') \in \mathrm{Sign}^{\mathcal{HI}^*}$ and $(M', W') \in \mathrm{Mod}^{\mathcal{HI}^*}(\Sigma', \mathrm{Nom}', \Lambda')$. Then for any $\rho \in \mathrm{Sen}^{\mathcal{HI}^*}(\Sigma, \mathrm{Nom}, \Lambda)$ and $w \in |W|(= |W'|)$,*

$$\mathrm{Mod}^{\mathcal{HI}^*}(\varphi)(M', W') \models^w \rho \text{ iff } (M', W') \models^w \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(\rho)$$

Proof. Let us denote $\mathrm{Mod}^{\mathcal{HI}^*}(\varphi)(M', W')$ by (M, W).

We start with the observation that $W'_{\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a)} = W_a$. The result follows by induction over the structure of the modalities. The base case, when $a \in \Lambda_1$, comes from the reduct definition in $\mathcal{HI}^*$. For $a = a_1; a_2$ we reason

$$\begin{array}{ll}
 & W'_{\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_1;a_2)} \\
= & \quad \{ \text{ defn. of } \mathrm{Sen}^{\mathcal{HI}^*} \} \\
 & W'_{\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_1);\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_2)} \\
= & \quad \{ \text{ defn. of actions interpretation} \} \\
 & W'_{\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_1)}; W'_{\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a_2)} \\
= & \quad \{ \text{induction hypothesis} \} \\
 & W_{a_1}; W_{a_2} \\
= & \quad \{ \text{ defn. of actions interpretation} \} \\
 & W_{a_1;a_2}
\end{array}$$

and the remaining cases are analogous.

For any $w \in |W|(= |W'|)$,

$$\begin{array}{ll}
 & (M, W) \models^w \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(\langle a \rangle \rho) \\
\Leftrightarrow & \quad \{ \text{ defn. of } \mathrm{Sen}^{\mathcal{HI}^*} \} \\
 & (M, W) \models^w \langle \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a) \rangle \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(\rho) \\
\Leftrightarrow & \quad \{ \text{defn. of } \models \} \\
 & (M, W) \models^{w'} \mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(\rho), \text{ for some } (w, w') \in W_{\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a)} \\
\Leftrightarrow & \quad \{ \text{induction hypothesis + } W'_{\mathrm{Sen}^{\mathcal{HI}^*}(\varphi)(a)} = W_a \} \\
 & (M', W') \models^{w'} \rho, \text{ for some } (w, w') \in W'_a \\
\Leftrightarrow & \quad \{ \text{defn of } \models \} \\
 & (M', W') \models^w \langle a \rangle \rho
\end{array}$$

The proof for $[a]\rho$ is similar. The remaining cases are taken from the the satisfaction condition of $\mathcal{HI}$. □

Therefore $\mathcal{HI}^*$ is an institution:

Corollary 5.1. *Let $\mathcal{HI}^*$ be the dynamic extension of the hybridisation of an institution $\mathcal{I}$, $\varphi : \Delta \to \Delta' \in \mathrm{Sign}^{\mathcal{HI}^*}$, $(M', W') \in \mathrm{Mod}^{\mathcal{HI}^*}(\Delta')$. Then, for any $\rho \in \mathrm{Sen}^{\mathcal{HI}^*}(\Delta')$,*

$$\mathrm{Mod}^{\mathcal{HI}^*}(\varphi)(M', W') \models^{\mathcal{HI}^*}_{(\Sigma,\mathrm{Nom},\Lambda)} \rho \textit{ iff } (M', W') \models^{\mathcal{HI}^*}_{(\Sigma',\mathrm{Nom}',\Lambda')} \mathrm{Sen}^{\mathcal{HI}}(\varphi)(\rho)\,.$$

Example: *The* IIP *device*

The safe behaviour of a IIP device also depends on its ability to allow bolus administration only in the expected states. More precisely, when not in *Basal* or *TBasal*, paths starting in the initial state and ending in *bolus* do not exist. A proof for this property may start by imposing the unreachability of *bolus* through bol_c,

$$\langle bol_c \rangle bolus \Rightarrow (Basal \vee TBasal)\,,$$

which is promptly verified in the HETS platform. Such condition is, however, too strong. Actually, it entails our goal, but the converse does not happen. A weaker statement is the property that the initial state does not directly reaches *bolus* through bol_c when not in *Basal* or *TBasal*,

$$\neg(Basal \vee TBasal) \Rightarrow \neg(\langle bol_c \rangle bolus \vee bolus)\,,$$

but this property is not strong enough to entail the goal. Using regular expressions as discussed above leads to a suitable formalisation of this property:

$$\neg(Basal \vee TBasal) \Rightarrow \neg\langle bol_c^* \rangle bolus$$

or more generally,

$$\neg(Basal \vee TBasal) \Rightarrow \neg\langle (res_c \cup diag_c \cup sus_c \cup bol_c)^* \rangle bolus\,.$$

As expected, this enrichment makes also possible to express liveness constraints. Such is the case of the requirement stated above as a motivation for this section, expressed as $\langle sus_c^* \rangle\, sus$.

As a final example, consider the $\mathcal{H}^2$TL sentence stating that *Fail*'s initial state has no outgoing transitions to *sus* or *nor*,

$$@_{Fail}\neg(\langle sus_c \rangle\,(sus \vee nor) \vee \langle diag_c \rangle\,(sus \vee nor) \vee \langle bol_c \rangle(sus \vee nor) \vee \langle res_c \rangle(sus \vee nor))\,,$$

which may be simplified as follows

$$@_{Fail}\neg(\langle sus_c \cup diag_c \cup bol_c \cup res_c \rangle(sus \vee nor))\,.$$

---

6. Conclusions

6.1. What was achieved

This paper introduced a rigorous and flexible method for the specification of reconfigurable systems based on hybrid logics generated on top of whatever logic is found adequate to describe the system's local configurations. This shows how a logic construction which is of largely theoretical interest — the *hybridisation* process — has an effective and novel application in handling a non trivial problem in Software Engineering.

Actually, reconfigurability is, at present, more the norm than the exception in software design. The method was illustrated in detail with the description of an insulin infusion pump (the IIP device), which provides a small, but clear example of a reconfigurable system. Another typical, everyday example is offered by cloud based applications that elastically react to clients demands.

On concluding, the main distinguished features of this work can be summarised as follows:

- First of all the introduction of hybrid features on top of the modal language used to specify the overall transition structure of a reconfigurable system, makes it possible to refer to individual configurations in an explicit way, leading to more flexible and precise specifications. For example, nominals and the corresponding satisfaction operators give the specifier a "surgical" precision in talking about the system's configurations.

- On the other hand, through hybridisation of each logic selected to specify individual configurations, we get a single, powerful logic weaving together local and global aspects to reason about the system. The two corresponding conceptual levels get unified through the use of a common logic which bears in its own structure local and global means of expressiveness.

- Finally, the institution-based construction used provides a precise way to transport specifications and proofs from one logic to another that can be realised in a formal tool for the proposed method.

In the sequel the paper contribution is compared to related work and, finally, current research directions are reported.

6.2. Related work

As stated, we intend to explore hybridised logics to frame a general approach to the specification of reconfigurable systems. By *general* we mean independent of whatever logic one finds suitable to describe the system's individual configurations. The *rationale* underlying our approach seeks to combine two basic dimensions in systems specification: one which emphasises *behaviour* and its evolution, another focused on *data* and their transformations. To be able to cope, within a single formalism, with both data structuring and prescription of functionality, as well as with specification and analysis of (externally observable) behaviour remains a main challenge for Software Engineering.

Behaviour is typically specified through (some variant of) *state machines*. Such models capture the system's evolution in terms of event occurrences and their impact in the system's internal state configuration (see e.g. [1]). Data types and services upon them, on the other hand, are often presented as theories in suitable logics, over a signature which offers a syntactic interface to the system. Semantics is, then, given by a class of concrete algebras acting as models of the specified theory (see e.g. [83]).

Our starting point is that these two dimensions are interconnected: the functionality offered by a reconfigurable system, at each moment, may depend on the stage of its evolution. In [71] the reconfiguration dynamics is modelled as a transition system, whose nodes are interpreted as the different configurations it may assume. Therefore, each of such nodes is endowed with an algebra, or even a first-order structure, to formally characterise the semantics of the services offered in the corresponding configuration. Technically, models of reconfigurable systems are given as *structured* state-machines whose states denote *algebras*, rather than *sets*. Structured transition systems [38] are, therefore, the semantic structure underlying the approach proposed. They are usually obtained by extending the bare structure of (sets of) states and transitions with further elements in either of them (e.g., structured labels, weights, functions, algebraic structure on states, etc). A quite general characterisation was proposed by R. Heckel and A. Corradini through the concept of *lax coalgebra* [37] which incorporates algebraic structure in both labels and states.

There are two ways to mathematically represent transition systems: either as *graphs* or as *coalgebras* by regarding the transition relation as a function from states to some collection of states whose shape is determined by a suitable endofunctor. It is not surprising that the literature on formal models of software reconfiguration explores both these paths. Graph rewriting techniques [95], notably the double pushout approach [58], have been extensibly used in modelling the evolution of dynamic systems. Typical applications emerge in the areas of mobile processes, from the π-calculus [81] and its variants [96] to the latter work of Robin Milner in bigraphs [36, 82], architectural evolution [108, 25] or coordination of software services [67], among many others.

Representing reconfiguration as transition systems described by coalgebras or, more often, in terms of their relational counterpart, has also been considered in the literature. The approaches are more diverse than in the graph-oriented trend and often endowed with a particular variant of a modal logic. Some examples include the separate specification of a second transition structure to monitor the basic one, often called a reconfiguration manager [10], the use of feature enriched transition systems [34] or well-structured transition systems [51]. The latter incorporates an order on (an infinite) state space, compatible with the transitions, with interesting decidability results. The specification of contracts or interaction conflicts in either states or transitions, in the context of the well known design-by-contract formalism, also provides mechanisms to talk about reconfigurations (see e.g. [102, 49, 11, 39]).

In both cases a critical ingredient to incorporate the reconfigurability dimension in specifications is the ability to add structure, typically algebraic structure, to the transition system modelling the system's behaviour. In a sense such is also the path taken in this work. But the combination of what, after Rutten's seminal work on universal coalgebra, are called algebraic and coalgebraic structures, has a long trace in Computer Science.

A first landmark was the whole research trend on *behavioural satisfaction*, early references being [52] and [93]. Hidden-sort algebra [54], embodying a fundamental distinction between visible values and internal states, which can only be observed in an indirect way, is an example of a behavioural formalism whose development was triggered, from the outset, by research on the foundations of object-oriented programming. Specification with *coherent hidden algebras* [45], *observational algebras* [60] and *behavioural reasoning* [75] are remarkable approaches in this line. In [97], the authors provide a comprehensive account of the area.

Another research direction, somehow closer to the approach proposed in this work, seeks to combine explicitly algebraic specifications with state-based structures. Also motivated by the emergence of object orientation in the 80's, the specification language TROOL [65] is a paradigmatic example. Objects, defined by attributes and evolving in response to events, are described as abstract data types and their evolution as linear processes specified in a temporal logic. A logic which combines many-sorted first-order logic with branching-time combinators, with both initial and loose semantics is introduced by Costa in [40].

Introduced by M. Broy and M. Wirsing in [23], *algebraic state machines* take algebraic specifications as states, an idea which was also present in Y. Gurevich seminal work on *evolving algebras* [57], posteriorly renamed to *abstract states machines* [21]. These machines, aiming at modelling arbitrary computational processes, consists of transition systems where each state has a structure of an algebra. The initial state consists of a particular algebra and each transition in a command that triggers an update on the current algebra. Hence, the set of transitions can be regarded as an abstract (imperative) program to be executed over the assigned initial state. The impact of abstract state machines in formal modelling cannot be understated. Several key ideas were borrowed by, and later incorporated in, popular model-oriented formalisms, namely the B method. A recent manifestation of this *states as algebras* perspective appears in the work of M. Bidoit and R. Hennicker in [15] as a semantic foundation for the contract-based design of software components. This perspective developed into a whole approach to software architecture based on a two layered semantics (at the interface and internal levels) with precise notions of composition and refinement.

The perspective of the presented specification method has several points of contact with these approaches based on structured transition systems. Note, however, that in our models a state does not correspond to a configuration of variables over a unique, common, fixed first-order structure, but to a specific structure modelling the configuration behaviour and functionality. Technically, we resort to rigid variables for non rigid operations, in contrast to other, more disseminated approaches where rigid operations act upon non rigid variables. On the other hand our specifications are always axiomatic and expressed in a logic which results from the hybridisation of the logic found suitable for each application to capture its possible configurations.

The use of different, often domain-tailored logics to specify reconfigurations in software constitutes a wide and heterogenous landscape in which our own approach fits in. We mention some examples, for illustration but with no pretension of exhaustibility. An important one is J. Meseguer's rewriting logic [79] and its MAUDE realisation, a language whose dynamics is based on the concurrent transformation of a 'soup' of objects and messages. A detailed overview, including references to modelling evolution and self-adaptability, is provided in [80]. Logic based formalisms are also common in specifying reconfiguration in component-based paradigms, dynamic software architectures and coordination schemes. An example of the first is given by the work of O. Kouchnarenko and her collaborators in which reconfigurations are specified in a temporal pattern logic [48] in the context FRACTAL [24], a paradigmatic component model. The work of T. Maibaum [2, 30] and J. Fiadeiro [50], as well as of the Pisa or the Munchen Schools (see e.g. [26] or [106], the latter work developed in an institutional framework), among many others, exemplify applications to dynamic architectures [113]. Finally, on the coordination side, the work of D. Clarke on what is called *reconfiguration logic* [33] expresses evolution of REO [4] connectors in formulas of a modal logic evaluated over constrained automata.

6.3. Future work

Current work aims at addressing some limitations of the method proposed here and extending its scope of application. A known limitation concerns interface reconfiguration, *i.e.*, the possibility of different (local) configurations being specified over different signatures. A solution exists for specifications in hybrid(ised) equational logic as discussed in reference [74]. However, addressing the general case remains an open issue. In an orthogonal directions lies the possibility of resorting to different logics to model different configurations, a topic which can be explored by taking the hybridisation of a product of the corresponding institutions with a suitable choice of operators.

Although the paper is focused in the specification stage of software development, current work includes the study of both a corresponding refinement process [72], and a requirements analysis stage. Actually, the design of comprehensive pallet of boilerplates for requirements elicitation of reconfigurable systems, illustrated in Section 3 is a relevant issue for the working software engineer. At present a HASKELL processor for these boilerplates is already available as a first step towards the definition of a *domain specific language* for this area of software technology. The combination of different sets of requirements expressed in hybridised versions $\mathcal{HI}$ of different base logics $\mathcal{I}$ is a challenge to explore.

The hybridisation process [76, 70], which, underlies the method discussed here, is rather flexible and able to cope, as discussed in Section 3, with several forms of quantification (*e.g.*, of nominals, modalities, etc.) which bring out remarkable expressive power. In some cases, as one would expect, this may rule out the existence of suitable first order encodings for the logic, thus reducing the existent tool support for the method. Computational support for reasoning about hybrid specifications is crucial from the Software Engineering point of view, which makes us to pay careful attention to this issue [88]. Encodings to second-order-logic are also being developed.

One can go even further by also taking as a parameter the logic used for the reconfiguration stage. This entails the need for developing a general method for asymmetric combination of logics parametric not only on the base logic used to express local requirements, but also on the *shape* of the transition structure. The latter, understood as a coalgebra, can be typed by a given functor. This *coalgebraising* process will go a step ahead of what is currently done (e.g. in hybridisation or temporalisation of logics), providing not only a more general, flexible approach, but also bringing upfront the whole coalgebraic machinery to the design of requirement-driven specification logics. This is left to future work, but note that a similar observation was already made by C. Kupke and D. Pattinson [68]. In their work, however, the emphasis is put on exploring different transition semantics, rather than on the combination of the latter with an arbitrary logic, i.e. the base logic remains fixed.

As clarified above, the notion of a reconfiguration is understood here as a statically programmed change of operation mode. This covers situations like the deletion of a component or a failure in a communication link which can be foreseen at design time and planned for by then using, for example, a fault tolerance scheme. Truly unplanned dynamic reconfigurations, involving, for example, the dynamic plugging of a new component enforced by the system's environment would require complementary mechanisms [66]. One may therefore question the advantages of the method proposed in this paper in comparison to a notationally simpler approach in which state changes are captured by "flattening" the reconfiguration structure as another local dimension of the (global) state. We believe our approach to be methodologically simpler, avoiding to mix mode information with other state dimensions and directly encoding reconfigurations as transitions between modes. In some cases, moreover, it may lead to simpler verification strategies. Actually, references [88, 89] show that the hybridisation method can be extended so that not only the logic is hybridised but also its calculus is systematically enriched into a calculus for the hybridised logic. The latter is shown to be sound and complete whenever the calculus associated to the underlying, base logic is. Reasoning at this level may thus pay back as an alternative to flattening the whole structure and having to resort to, e.g., first order setting to capture the whole picture.

If reconfigurations increase the availability and the reliability of software systems by allowing their architectures to evolve at run-time, there is a number of other domains in Computing where reconfigurability, and the search for suitable formal methods, are emerging as a crucial issue. These includes areas in the intersection of discrete and continuous behaviour, namely in what concerns *sensor networks* [106] and *robotics* [110]. Whether the method introduced in this paper scales to those domains, as well as to real, industrial cases, on the other hand, only time and effort will tell. Therefore, the study of hybridisation of logics to deal with probabilistic [35, 47, 59] and continuous [91] reasoning appears as the next, natural step to take.

Acknowledgments.

This work is financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within projects POCI-01-0145-FEDER-016692 and UID/MAT/04106/2013. The first author is further supported by the BPD FCT grant SFRH/BPD/103004/2014, and R. Neves is sponsored by FCT grant SFRH/BD/52234/2013. M. A. Martins is also funded by the EU FP7 Marie Curie PIRSESGA-2012-318986 project GeTFun: Generalizing Truth-Functionality.

References

[1] Luca Aceto, Anna Ingólfdótir, Kim G. Larsen, and Jiri Srba. *Reactive Systems: Modelling, specification and verification*. Cambridge University Press, 2007.

[2] Nazareno Aguirre and Tom S. E. Maibaum. A temporal logic approach to the specification of reconfigurable component-based systems. In *17th IEEE Intern. Conf. on Automated Software Engineering (ASE 2002, 23-27 September 2002, Edinburgh, Scotland, UK)*, pages 271–274, 2002.

[3] Jaume Agust-Cullell, Francesc Esteva, Pere Garcia, and Lluis Godo. Formalizing multiple-valued logics as institutions. In B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, editors, *3rd Intern. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 90, Paris, France, July 2-6, 1990)*, volume 521 of *Lecture Notes in Computer Science*, pages 269–278. Springer, 1990.

[4] Farhad Arbab. Reo: a channel–based coordination model for component composition. *Mathematical Structures in Comp. Sci.*, 14(3):329–366, 2004.

[5] Carlos Areces and Patrick Blackburn. Bringing them all together. *J. Log. Comput.*, Oxford Journals, 11(5):657–669, 2001.

[6] C. Areces and B. ten Cate. Hybrid logics. In P. Blackburn, F. Wolter, and J. van Benthem, editors, *Handbook of Modal Logic*, Studies in Logic and Practical Reasoning (volume 3), pages 822–868. Elsevier, 2007.

[7] D. Arney, R. Jetley, P. Jones, Insup Lee, and O. Sokolsky. Formal methods based development of a pca infusion pump reference model: Generic infusion pump (gip) project. In *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007. HCMDSS-MDPnP. Joint Workshop on*, pages 23–33, 2007.

[8] Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: the common algebraic specification language. *Theor. Comput. Sci.*, 286(2):153–196, 2002.

[9] Pedro Baltazar. Probabilization of logics: Completeness and decidability. *Logica Universalis*, pages 1–38, 2013.

[10] Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In D. Giannakopoulou and D. Mery, editors, *Formal Methods (FM-12, Paris, France, August 27-31, 2012)*, volume 7436 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 2012.

[11] Sebastian S. Bauer, Rolf Hennicker, and Martin Wirsing. Interface theories for concurrency and data. *Theor. Comput. Sci.*, 412(28):3101–3121, 2011.

[12] Peter Baumgartner, Björn Pelzer, and Cesare Tinelli. Model evolution with equality - revised and implemented. *J. Symb. Comput.*, 47(9):1011–1045, 2012.

[13] Christoph Beierle and Gabriele Kern-Isberner. Looking at probabilistic conditionals from an institutional point of view. In G. Kern-Isberner, W. Rödder, and F. Kulmann, editors, *Conditionals, Information, and Inference (Revised Selected Papers of WCII 2002, Hagen, Germany, May 13-15, 2002)*, volume 3301 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2005.

[14] Michel Bidoit and Rolf Hennicker. Constructor-based observational logic. *J. Log. Algebr. Program.*, 67(1-2):3–51, 2006.

[15] Michel Bidoit and Rolf Hennicker. An algebraic semantics for contract-based software components. In J. Meseguer and G. Rosu, editors, *Algebraic Methodology and Software Technology (AMAST 2008 - Urbana, IL, USA, July 28-31, 2008*, volume 5140 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 2008.

[16] Dines Björner and Martin Henson (eds). *Logics of Specification Languages*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2008.

[17] Patrick Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of IGPL*, 8(3):339–365, 2000.

[18] Patrick Blackburn. Arthur Prior and hybrid logic. *Synthese*, 150(3):329–372, 2006.

[19] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Number 53 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.

[20] Thomas Bolander and Torben Braüner. Tableau-based decision procedures for hybrid logic. *J. Log. Comput.*, 16(6):737–763, 2006.

[21] Egon Börger and Robert F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.

[22] Torben Braüner. *Hybrid Logic and its Proof-Theory*. Applied Logic Series. Springer, 2010.

[23] Manfred Broy and Martin Wirsing. Algebraic state machines. In T. Rus, editor, *Algebraic Methodology and Software Technology (AMAST 2000, Iowa City, Iowa, USA, May 20-27, 2000)*, volume 1816 of *Lecture Notes in Computer Science*, pages 89–188. Springer, 2000.

[24] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The Fractal component model and its support in java. *Softw., Pract. Exper.*, 36(11-12):1257–1284, 2006.

[25] Roberto Bruni, Antonio Bucchiarone, Stefania Gnesi, Dan Hirsch, and Alberto Lluch-Lafuente. Graph-based design and analysis of dynamic software architectures. In P. Degano, R. De Nicola, and J. Meseguer, editors, *Concurrency, Graphs and Models (Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday)*, volume 5065 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 2008.

[26] Roberto Bruni, Alberto Lluch-Lafuente, and Ugo Montanari. Style-based architectural reconfigurations. *Bulletin of the EATCS*, 94:161–180, 2008.

[27] Rod Burstall and Răzvan Diaconescu. Hiding and behaviour: an institutional approach. In William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 75–92. Prentice-Hall, 1994.

[28] Rod M. Burstall and Joseph A. Goguen. The semantics of CLEAR, a specification language. In D. Bjørner, editor, *Abstract Software Specifications (1979 Copenhagen Winter School, January 22 - February 2, 1979)*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332. Springer, 1980.

[29] Carlos Caleiro, Paulo Mateus, Amílcar Sernadas, and Cristina Sernadas. Quantum institutions. In K. Futatsugi, J.-P. Jouannaud, and J. Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2006.

[30] Pablo F. Castro, Nazareno Aguirre, Carlos Gustavo Lopez Pombo, and T. S. E. Maibaum. Towards managing dynamic reconfiguration of software systems in a categorical setting. In A.Cavalcanti, D. Deharbe, M.-C. Gaudel, and J. Woodcock, editors, *Theoretical Aspects of Computing (ICTAC 2010, Natal, Rio Grande do Norte, Brazil, September 1-3, 2010*, volume 6255 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2010.

[31] Balder ten Cate and Massimo Franceschet. On the complexity of hybrid logics with binders. In C.-H. Luke Ong, editor, *Computer Science* Logic (CSL 2005 - Oxford, UK, August 22-25, 2005), volume 3634 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2005.

[32] Corina Cîrstea. An institution of modal logics for coalgebras. *J. Log. Algebr. Program.*, 67(1-2):87–113, 2006.

[33] Dave Clarke. A basic logic for reasoning about connector reconfiguration. *Fundam. Inform.*, 82(4):361–390, 2008.

[34] Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-Francois Raskin. Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking. *IEEE Transactions on Software Engineering*, 99(PrePrints):1, 2013.

[35] Giulianella Coleti and Romano Scozzafava. *Probabilistic Logic in a Coherent Setting*. Trends in Logic, Studia Logica Library (volume 15). Kluwer Academic Publishers, 2002.

[36] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Static BiLog: a unifying language for spatial structures. In W. Penczek and G. Rozenberg, editors, *Half a century of inspirational research, honouring the scientific influence of Antoni Mazurkiewicz*, volume 80 (1–3) of *Fundamenta Informaticae*, pages 91–110. IOS Press, 2007.

[37] Andrea Corradini, Martin Große-Rhode, and Reiko Heckel. A coalgebraic presentation of structured transition systems. *Theor. Comput. Sci.*, 260(1-2):27–55, 2001.

[38] Andrea Corradini and Ugo Montanari. An algebraic semantics for structured transition systems and its applications to logic programs. *Theor. Comput. Sci.*, 103(1):51–106, 1992.

[39] Roberto Di Cosmo, Stefano Zacchiroli, and Gianluigi Zavattaro. Towards a formal component model for the cloud. In G. Eleftherakis, M. Hinchey, and M. Holcombe, editors, *Software Engineering and Formal Methods (SEFM 2012, Thessaloniki, Greece, October 1-5, 2012)*, volume 7504 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2012.

[40] Gerardo Costa and Gianna Reggio. Specification of abstract dynamic-data types: A temporal logic approach. *Theor. Comput. Sci.*, 173(2):513–554, 1997.

[41] Răzvan Diaconescu. *Institution-independent Model Theory*. Studies in Universal Logic. Birkhäuser Basel, 2008.

[42] Răzvan Diaconescu. Quasi-boolean encodings and conditionals in algebraic specification. *J. Log. Algebr. Program.*, 79(2):174–188, 2010.

[43] Răzvan Diaconescu. On quasi-varieties of multiple valued logic models. *Math. Log. Q.*, 57(2):194–203, 2011.

[44] Răzvan Diaconescu. Quasi-varieties and initial semantics in hybridized institutions. *Journal of Logic and Computation DOI:10.1093/logcom/ext016*, 2013.

[45] Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. *Theor. Comput. Sci.*, 285:289–318, 2002.

[46] Răzvan Diaconescu and Alexandre Madeira. Encoding hybridized institutions into first order logic. *Math. Struct. in Comput. Sci.*, 26(5): 745-788, 2016.

[47] Ernst-Erich Doberkart. *Stochastic Coalgebraic Logic*. Monographs in Theoretical Computer Science, an EATCS Series. Springer, 2010.

[48] Julien Dormoy, Olga Kouchnarenko, and Arnaud Lanoix. Using temporal logic for dynamic reconfigurations of components. In L. S. Barbosa and M. Lumpe, editors, *Formal Aspects of Component Software (Revised Selected Papers of FACS 2010, Guimaraes, Portugal, October 14-16, 2010)*, volume 6921 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 2010.

[49] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[50] Jose Luiz Fiadeiro and Antonia Lopes. A model for dynamic reconfiguration in service-oriented architectures. *Software and System Modeling*, 12(2):349–367, 2013.

[51] Alain Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[52] V. Giarratana, F. Gimona, and Ugo Montanari. Observability concepts in abstract data type specifications. In A. W. Mazurkiewicz, editor, *Mathematical Foundations of Computer Science (MFCS, Gdansk, Poland, September 6-10, 1976)*, volume 45 of *Lecture Notes in Computer Science*, pages 576–587. Springer, 1976.

[53] Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *J. ACM*, 39(1):95–146, 1992.

[54] Joseph A. Goguen and Grant Malcolm. A hidden agenda. *Theor. Comput. Sci.*, 245(1):55–101, 2000.

[55] Siegfried Gottwald. *A Treatise on Many-Valued Logics*. Studies in Logic and Computation (volume 9). Research Studies Press, 2001.

[56] Daniel Götzmann, Mark Kaminski, and Gert Smolka. Spartacus: A tableau prover for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 262:127–139, 2010.

[57] Yuri Gurevich. Evolving algebras. In *IFIP Congress (1)*, pages 423–427, 1994.

[58] Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer*

Science, 11(5):637–688, 2001.

[59] Rolf Haenni, Jan-Willem Romeijn, Gregory Wheeler, and Jon Williamsont. *Probabilistic Logics and Probabilistic Networks*. Synthese Library: Studies in Epistemology, Logic, Methodology, and Philosophy of Science (volume 350). Springer, 2011.

[60] Rolf Hennicker and Michel Bidoit. Observational logic. In Armando Martin Haeberer, editor, *AMAST*, volume 1548 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1998.

[61] Christian Hoareau and Ichiro Satoh. Hybrid logics and model checking: A recipe for query processing in location-aware environments. In *22nd Intern. Conf. on Advanced Information Networking and Applications (AINA 2008, GinoWan, Okinawa, Japan, March 25-28, 2008)*, pages 130–137. IEEE Computer Society, 2008.

[62] Wilfrid Hodges. *Model Theory*. Cambridge University Press, 1993.

[63] Guillaume Hoffmann and Carlos Areces. Htab: a terminating tableaux system for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 231:3–19, 2009.

[64] Andrzej Indrzejczak. Modal hybrid logic. *Logic and Logical Philosophy*, 16:147–257, 2007.

[65] Ralf Jungclaus, Gunter Saake, Thorsten Hartmann, and Cristina Sernadas. TROLL - a language for object-oriented specification of information systems. *ACM Trans. Inf. Syst.*, 14(2):175–211, 1996.

[66] G. Karsai, F. Massacci, L. J. Osterweil, and I. Schieferdecker. Evolving embedded systems.. *IEEE Computer*, 43(5):34–40, 2010.

[67] Christian Krause, Ziyan Maraikar, Alexander Lazovik, and Farhad Arbab. Modeling dynamic reconfigurations in reo using high-level replacement systems. *Sci. Comput. Program.*, 76(1):23–36, 2011.

[68] Clemens Kupke and Dirk Pattinson. Coalgebraic semantics of modal logics: An overview. *Theor. Comput. Sci.*, 412(38):5070–5094, 2011.

[69] Martin Lange. Model checking for hybrid logic. *J. of Logic, Lang. and Inf.*, 18(4):465–491, 2009.

[70] Alexandre Madeira. *Foundations and techniques for software reconfigurability*. PhD thesis, Universidades do Minho, Aveiro and Porto (Joint MAP-i Doctoral Programme), July 2013.

[71] Alexandre Madeira, José M. Faria, Manuel A. Martins, and Luís Soares Barbosa. Hybrid specification of reactive systems: An institutional approach. In G. Barthe, A. Pardo, and G. Schneider, editors, *Software Engineering and Formal Methods (SEFM 2011, Montevideo, Uruguay, November 14-18, 2011)*, volume 7041 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 2011.

[72] Alexandre Madeira, Manuel A. Martins, and Luís S. Barbosa. Bisimilarity and refinement for hybrid(ised) logics. In John Derrick, Eerke A. Boiten, and Steve Reeves, editors, *Refine - Proceedings 16th International Refinement Workshop*, volume 115 of *Electronic Proceedings in Theoretical Computer Science*, pages 84–98, 2013.

[73] Alexandre Madeira, Manuel A. Martins, and Luís Soares Barbosa. Boilerplates for reconfigurable systems: A language and its semantics. In André Rauber Du Bois and Phil Trinder, editors, *SBLP*, volume 8129 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 2013.

[74] Alexandre Madeira, Renato Neves, Manuel A. Martins, and Luís Soares Barbosa. When even the interface evolves... In *Proc. TASE*, pages 79–82. IEEE, 2013.

[75] Manuel A. Martins and Don Pigozzi. Behavioural reasoning for conditional equations. *Mathematical Structures in Comp. Sci.*, 17(5):1075–1113, 2007.

[76] Manuel A. Martins, Alexandre Madeira, Răzvan Diaconescu, and Luís Soares Barbosa. Hybridization of institutions. In A. Corradini, B. Klin, and C. Cîrstea, editors, *Algebra and Coalgebra in Computer Science (CALCO 2011, Winchester, UK, August 30 - September 2, 2011)*, volume 6859 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2011.

[77] Paolo Masci, Anaheed Ayoub, Paul Curzon, Insup Lee, Oleg Sokolsky, and Harold Thimbleby. Model-based development of the generic pca infusion pump user interface prototype in pvs. In Friedemann Bitsch, Jrmie Guiochet, and Mohamed Kaniche, editors, *Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 228–240. Springer Berlin Heidelberg, 2013.

[78] Radu Mateescu and Mihaela Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *Sci. Comput. Program.*, 46(3):255–281, 2003.

[79] Jose Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

[80] Jose Meseguer. Twenty years of rewriting logic. *J. Log. Algebr. Program.*, 81(7-8):721–781, 2012.

[81] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1):1–77, 1992.

[82] Robin Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[83] Till Mossakowski, Anne Haxthausen, Donald Sannella, and Andrzej Tarlecki. CASL: The common algebraic specification language: Semantics and proof theory. *Computing and Informatics*, 22:285–321, 2003.

[84] Till Mossakowski, Christian Maeder, Mihai Codescu, and Dominik Lucke. HETS User Guide - Version 0.99. Technical report, DFKI Lab Bremen, April 2013.

[85] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, Hets. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007 - Braga, Portugal, March 24 - April 1, 2007)*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer, 2007.

[86] Till Mossakowski and Markus Roggenbach. Structured CSP - a process algebra as an institution. In J. L. Fiadeiro and P.-Y. Schobbens, editors, *Recent Trends in Algebraic Development Techniques (Revised Selected Papers of WADT 2006, La Roche en Ardenne, Belgium, June 1-3, 2006)*, volume 4409 of *Lecture Notes in Computer Science*, pages 92–110. Springer, 2006.

[87] Renato Neves, Alexandre Madeira, Manuel A. Martins, and Luís Soares Barbosa. Hybridisation at work. In Reiko Heckel and Stefan Milius, editors, *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings*, volume 8089 of *Lecture Notes in Computer Science*, pages 340–345, 2013.

[88] Renato Neves, Manuel A. Martins, and Luís Soares Barbosa. Completeness and decidability results for hybrid(ised) logics. In C. Braga and N. Martí-Oliet, editors, *Formal Methods: Foundations and Applications (Proc. Brazilian Symp. on Formal Methods, SBMF 2014)*, volume 8941 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 2015.

[89] Renato Neves, Alexandre Madeira, Manuel A. Martins, and Luís Soares Barbosa. Proof Thery for Hybridi(sed) Logics. *Science of Computer Programming*, doi:10.1016/j.scico.2016.03.001, Elsevier (in print).

[90] Solomon Passy and Tinko Tinchev. An essay in combinatory dynamic logic. *Inf. Comput.*, 93(2):263–332, 1991.

[91] Andre Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, 2010.

[92] Arthur N. Prior. *Past, Present and Future*. Oxford University Press. 1967.

[93] Horst Reichel. Behavioral program specification. In David H. Pitt, Samson Abramsky, Axel Poigné, and David E. Rydeheard, editors, *CTCS*, volume 240 of *Lecture Notes in Computer Science*, pages 390–411. Springer, 1985.

[94] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2-3):91–110, 2002.

[95] Gregori Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol 1: Foundations*. World Scientific, 1997.

[96] Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2003.

[97] Donald Sannella and Andrzej Tarlecki. Observability concepts in abstract data type specification, 30 years later. In P. Degano, R. De Nicola, and J. Meseguer, editors, *Concurrency, Graphs and Model (Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday)*, volume 5065 of *Lecture Notes in Computer Science*, pages 593–617. Springer, 2008.

[98] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs on Theoretical Computer Science, an EATCS Series. Springer, 2012.

[99] Lutz Schröder and Till Mossakowski. HasCasl: Integrated higher-order specification and program development. *Theor. Comput. Sci.*, 410(12-13):1217–1260, 2009.

[100] Neeraj Kumar Singh, Hao Wang, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng. Report 18: Formalizing Insulin Pump using Event-B. Technical Report 18, McSCert, October 2014.

[101] Ian Sommerville. *Software Engineering*. Addison-Wesley, 9 edition, 2010.

[102] Gabriel Tamura, Rubby Casallas, Anthony Cleve, and Laurence Duchien. Qos contract-aware reconfiguration of component architectures using e-graphs. In *Formal Aspects of Component Software (Revised Selected Papers of FACS 2010, Guimaraes, Portugal, October 14-16, 2010)*, volume 6921 of *Lecture Notes in Computer Science*, pages 34–52. Springer, 2010.

[103] Andrzej Tarlecki. Abstract specification theory: an overview. In M. Broy and M. Pizka, editors, *Models, Algebras, and Logics of Engineering Software*, volume 191 of *NATO Science Series, Computer and Systems Sciences, VOL 191*, pages 43–79. IOS Press, 2003.

[104] Johan van Bentham. *Modal Logic and Classic Logic*. Humanities Press, 1983.

[105] Jan van Eijck. Hylotab-tableau-based theorem proving for hybrid logics. Technical report, CWI, Amsterdam, 2002.

[106] M. Birna van Riemsdijk, Rolf Hennicker, Martin Wirsing, and Andreas Schroeder. Service specification and matchmaking using description logic. In J. Meseguer and G. Roşu, editors, *Algebraic Methodology and Software Technology (AMAST 2008 - Urbana, IL, USA, July 28-31, 2008*, volume 5140 of *Lecture Notes in Computer Science*, pages 392–406. Springer, 2008.

[107] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In R. A. Schmidt, editor, *Automated Deduction (CADE-22, Montreal, Canada, August 2-7, 2009*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.

[108] Michel Wermelinger and José Luiz Fiadeiro. A graph transformation approach to software architecture reconfiguration. *Sci. Comput. Program.*, 44(2):133–155, 2002.

[109] Hao Xu and Tom Maibaum. An event-b approach to timing issues applied to the generic insulin infusion pump. In *Proceedings of the First international conference on Foundations of Health Informatics Engineering and Systems*, FHIES'11, pages 160–176, Berlin, Heidelberg, 2012. Springer-Verlag.

[110] Zheng Yu, Ian Warren, and Bruce A. MacDonald. Dynamic reconfiguration for robot software. In *IEEE Intern. Conf. on Automation Science and Engineering (CASE, Shanghai, China, 7-10 October 2006)*, pages 292–297. IEEE, 2006.

[111] Yi Zhang, Raoul Jetley, Paul L Jones, and Arnab Ray. Generic safety requirements for developing safe insulin pump software. *J Diabetes Sci Technol*, 5(6):1403–19, 2011.

[112] Yi Zhang, Paul L Jones, and Raoul Jetley. A hazard analysis for a generic insulin infusion pump. *J Diabetes Sci Technol*, 4(2):263–83, 2010.

[113] Zhikun Zhao and Wei Li. Specifying dynamic software architectures with dynamic description logic. *Jour. of Software*, 7(1):169–175, 2012.