Boilerplates for reconfigurable systems: a language and its semantics

Alexandre Madeira[1,2,3], Manuel A. Martins[2], Luís S. Barbosa[2]

[1] HASLab - INESC TEC and Universidade do Minho, Portugal
[2] CIDMA-Dep. of Mathematics, Universidade de Aveiro, Portugal
[3] Critical Software S.A., Portugal

Abstract. Boilerplates are simplified, normative English texts, intended to capture software requirements in a controlled way. This paper proposes a pallet of boilerplates as a requirements modelling language for reconfigurable systems, i.e., systems structured in different modes of execution among which they can dynamically commute. The language semantics is given as an hybrid logic, in an institutional setting. The mild use made of the theory of institutions, which, to a large extent, may be hidden from the working software engineer, not only provides a rigorous and generic semantics, but also paves the way to tool-supported validation.

1 Motivation and overview

Requirements Engineering [9] is the branch of software engineering concerned with the precise identification of goals and constraints of the services provided by systems. Typically, this involves understanding, modelling and documenting not only the needs of potential users or customers, but also the deployment contexts in which such systems under development will be used. The deliverable of this stage in the software development process must be expressed in a form that is amenable to analysis, communication, and subsequent implementation.

In practice requirements engineers start with ill-defined, often conflicting, ideas of what the new system is expected to do. They are supposed to make progress towards a detailed, technical specification of the system. This entails the need for suitable support methodologies to record and structure the relevant information, as well as to express it in a clear, easy to understand notation.

The notion of a *boilerplate*, first introduced in [9], is a step in this direction: for each class of requirements, within a specific domain, a generic template is defined so that capturing requirements amounts to instantiated well-characterized textual schemes written in simplified, normative English. Informally, a boilerplate is a standardized scheme that can be reused over and over again, and is amenable to some form of computer-based simulation. The term derives from steel manufacturing, where it refers to steel rolled into large plates for use in steam boilers. The intuition is that a boilerplate has been time-tested and is 'strong as steel' suitable for repeated reuse. The use of 'controlled natural language' for requirements elicitation is a successful practice in industry and, despite

of its informal character, does provide an interesting starting point towards more formal approaches.

Boilerplates are usually developed for specific business areas, classes of systems or typical design stages.This paper focus in *reconfigurable* systems. Those are systems whose form (i.e. resources involved, network topology, etc) changes along the computational process in response to varying context conditions.

The behavior of this kind of systems is indexed to a set of different run-time *configurations* between which the system commutes dynamically. Therefore, a specification takes the form of a *structured transition system*: transitions capture the evolution from one configuration to another, whereas each state corresponds to the full specification of data and services available at a particular configuration. Such local configurations can be described in different languages, ranging from, equational to first order logic or even to less conventional formalisms, e.g., fuzzy or multivalued logics. In the sequel we will refer to the logic used at the local level of configurations as the *base* logic.

If the base logic provides a language to express requirements relative to each configuration of the system, describing the reconfiguration dynamics itself requires a *modal* logic to express transition and change. Actually, we adopt an extension of ordinary modal logic in which dedicated propositional symbols, called *nominals*, each being true at exactly one possible state, are used to *name* states, i.e., the system's individual configurations. This extension is known as *hybrid* logic, whose roots go back Arthur N. Prior's work in the 1960s; see [1] for a detailed account and historic perspective. Along with nominals, it also introduces *satisfaction operators* $@_i\phi$, which formalise a statement ϕ being true at a specific configuration named i.

In such a context, the paper's contribution is twofold:

- first it introduces a collection of boilerplates for capturing typical requirements of reconfigurable systems;
- then, it takes seriously the challenge of providing a proper, unambiguous semantics for them.

Our perspective is that the methodological advantages of boilerplates, i.e. their conciseness and genericity, depends on the existence of a rigorous formal semantics for them, amenable to formal transformation and verification. On the other hand, the distinguishing feature of our approach is that boilerplates are parametric on whatever (*base*) logic is chosen for specifying the system's configurations.

The methodology proposed proceeds as follows: first a suitable base logic to express the properties of (local) configurations is chosen. Then, the requirements are collected into specific boilerplates which structure information on the relevant vocabulary, available configurations, events triggering reconfiguration and both local and global properties. Once instantiated, boilerplates are translated into specifications in (a suitable version of) *hybrid logic* (*e.g.* [2]) providing a formal description of requirements amenable to tool-supported validation. By the expression *'(suitable version of) hybrid logic'* we mean a language with enriches the base logic specific to each application with modalities and hybrid features to express reconfiguration and evolution. Such a language is derived in

a formal and systematic way — the so-called *hybridisation process* whose theory was developed by the authors in [13,4].

Going generic entails a price to pay: to seek for a suitably generic notion of logical system encompassing syntax, semantics and satisfaction. Fortunately the concept is already well-established in the so-called theory of institutions of Goguen and Burstall [6,3]. At expenses of some extra (and a bit heavy) notation, institutions offer an abstract representation of a logic, and their theory provides modular structuring and parameterization mechanisms which are defined 'once and for all', abstracting from the concrete particularities of the each specification logic [5]. The formal semantics for boilerplates proposed in this paper is framed in this setting: each logic (base and hybridised) is regarded as an *institution*.

Another advantage of the institutional framework is its ability to relate logics and transport results from one to another [14], which means that a theorem prover for the latter can be used to reason about specifications written in the former. Our approach takes advantage of this to provide 'for free' suitable tool support through a translation of collections of boilerplates to first-order logic and their validation in the HETS [16] tool.

The paper is organized as follows: Section 2 introduces a pallet of boilerplates for reconfigurable systems and illustrates their use through a small example. A formal semantics for this pallet of boilerplates is addressed in Section 3. Finally, Section 4 proposes a methodology for engineering requirements of reconfigurable systems, from their elicitation and expression in boilerplates until their validation and prototyping within the HETS framework.

The semantic framework used in the sequel is based on the theory of institutions and a method to generate hybrid from arbitrary logics. Part of it, namely the background formalism and notation, can be skipped at first reading without compromising a broader understanding of the paper's ideas. For the interested reader, details and examples are given in the Appendix.

2 A language of boilerplates for reconfigurability

As sketched in the previous section, requirements for reconfigurable systems are captured in a collection of boilerplates which, taken jointly, specify a structured transition system. Its states, corresponding to different *configurations*, or *modes of execution*, are endowed with a specific description of the functionality available locally. The boilerplates proposed below define globally the relevant modes of execution and the transition structure, as well as, at the local level of each mode, the interface of services available and their properties.

Basic boilerplates

Five classes of boilerplates are introduced to register requirements, structuring them as a (structured) transition system. The choice of the *base* logic $\mathcal{I}$ is made within the boilerplates concerned with the system's interface. A concrete instantiation of these boilerplates requires such a choice: notation $BP(\mathcal{I})$

stands therefore, for the set of boilerplates in which the requirements for local configurations are given in $\mathcal{I}$. The basic boilerplates proposed are as follows:

1. Identification of the relevant configurations:

   System plays the configurations <set of configurations>
   <Mode> *is a execution mode*

2. Definition of event sets able to trigger a mode transition, i.e., a system's reconfiguration:

   System has events <set of Event>
   <Event> *is an event*

3. Definition of the basic transition structure:

   System changes from <Mode> *to* <Mode> *through the event* <Event>
   System may change from <Mode> *to* <Mode> *through the event* <Event>

4. Definition of the system's interface:

   System interface is defined by <InterfaceExp>

5. Local specification, i.e., relative to the system's functionality at each configuration (stated in the chosen *base* logic):

   Property <Prop> *holds in all modes*
   Property <Prop> *holds in* <Mode>

6. Definition of possible transitions (i.e., reconfigurations) emerging from local properties (e.g., a certain limit value for a parameter is achieved).

   <Event> *changes modes satisfying* <Prop> *into modes satisfying* <Prop>
   <Event> *changes* <Mode> *to modes satisfying* <Prop>

An example

For example let us consider a small, self-contained example. Other examples appeared in the first author's PhD thesis [10]. For the moment, consider the following requirements for a quite peculiar, 'plastic' buffering structure:

> *A 'plastic' buffer is a versatile data structure with two distinct modes of execution: in one of them it behaves as a stack; in the other as a queue. The reconfiguration is triggered by by an external event 'shift'.*

We start fixing the transition structure between the buffer's (two) modes of execution.

Modes and events:

- **fifo** *is a mode*
- **lifo** *is a mode*
- **Shift** *is an event*

Transition structure:

System changes from <**lifo**> *to* <**fifo**> *through the event* <**shift**>
System changes from <**fifo**> *to* <**lifo**> *through the event* <**shift**>

For the specification of each execution mode, or configuration, one may resort to propositional logic $\mathcal{PL}$, the buffer requirements are expressed in $BP(\mathcal{PL})$. The following boilerplate fixes the local behaviour: the proposition *stack_bh* is to hold in configurations in which the buffer behaves like a stack; proposition *queu_bh* when it behaves as a queue.

System interface is defined by <**{stack_bh,queu_bh}**>

Hence,

- *Property queu_bh holds in* **fifo**
- *Property stack_bh holds in* **lifo**

In practice, however, the propositional setting may not be enough: most properties are better expressed in *equational* logic $\mathcal{EQ}$. Thus, one my state

System interface is defined by < $\mathbf{\Sigma_{Pbuffer}}$ >

where $\mathbf{\Sigma_{Pbuffer}}$ is the classical first-order signature of a stack/queue data type with *write*, *read* and *del* operations together with a constant *new* to denote the empty buffer. Hence, local properties are expressed by

- *Property* $\mathbf{read(write(m,e)) = e}$ *holds in* **lifo**
- *Property* $\mathbf{m = new \Rightarrow read(write(m,e)) = e}$ *holds in* **fifo**
- *Property* $\mathbf{\neg(m = new) \Rightarrow read(write(m,e)) = read(m)}$ *holds in* **fifo**
- *Property* $\mathbf{del(write(m,e)) = m}$ *holds in* **lifo**
- *Property* $\mathbf{\neg(m = new) \Rightarrow del(write(m,e)) = write(del(m),e)}$ *holds in* **fifo**
- *Property* $\mathbf{m = new \Rightarrow del(write(m,e)) = new}$ *holds in* **lifo, fifo**

A precise semantics for this sort of boilerplates is given in the following section by their transformation into a proper formal specifications in suitable hybrid logics.

3 A formal semantics for $BP(\mathcal{I})$

If the collection of boilerplates proposed here for reconfigurable systems leads naturally to models based on structured transition systems, the choice of (a variant of) hybrid logic for their semantics comes as no surprise. Reactive systems are classically expressed in modal languages; on the other hand, a naming mechanism for states makes easier to distinguish between properties valid in some, but not all, configurations.

The semantic framework is as follows: Once the system's requirements are captured in a collection $BP(\mathcal{I})$ of boilerplates instantiated over a *base* logic $\mathcal{I}$, its semantics is given by a systematic translation to a hybrid logic over $\mathcal{I}$. I.e, a logic whose language extends that of $\mathcal{I}$ with a set Λ of *modalities*, the corresponding *eventually* ($\langle\lambda\rangle$) and *henceforth* ($[\lambda]$) operators, for each $\lambda \in \Lambda$, a set Nom of *nominals* to name configurations, and, for each $i \in Nom$ a satisfaction operator $@_i$ enforcing the validity of its argument in configuration i. Formally, the collection of boilerplates gives rise to a proper specification in the hybrid

logic $\mathcal{HI}$ corresponding to $\mathcal{I}$. The generation of $\mathcal{HI}$ from $\mathcal{I}$, i.e., the *hybridisation* of $\mathcal{I}$, is also a systematic process whose technical details are summarised in the Appendix.

For the moment we shall concentrate in the process of generating a $\mathcal{HI}$-specification from a collection of boilerplates. Note the introduction of nominals to refer to local configurations and of modalities to state properties of the overall transition structure. This is better illustrated through an example. Let us, thus, revisit the buffer example.

In Section 2 two collections of boilerplates were considered for this example. The first one resorted to *propositional* logic $\mathcal{PL}$. Its semantics is, therefore, a generated specification in hybrid propositional logic $\mathcal{HPL}$:

spec ReconfBuffer1 =
nominal *fifo*, *lifo*
modalities *shift*
propositions *stack_bh*,*queue_bh*
• $@_{fifo} stack\_bh$
• $@_{lifo} queue\_bh$
• $@_{lifo} < shift > fifo$
• $@_{fifo} < shift > lifo$

The models M for this specification are standard Kripke structures. For instance, the structure defined over a set of two states $\{s_{lifo}, s_{fifo}\}$ and whose accessibility relation is $W_{shift} = \{(s_{lifo}, s_{fifo}), (s_{fifo}, s_{lifo})\}$. The value of propositions *stack_bh* and *queue_bh* is each state is as follows: $M_{s_{lifo}}(stack\_bh) = M_{s_{fifo}}(queue\_bh) = \top$ and $M_{s_{lifo}}(queue\_bh) = M_{s_{fifo}}(stack\_bh) = \bot$.

The second, richer set of boilerplates resorted to *equational* logic $\mathcal{EQ}$ to capture local requirements equationally. The resulting specification is now expressed in hybrid equational logic $\mathcal{HEQ}$, as follows.

spec ReconfBuffer2 =
nominal *fifo*, *lifo*
modalities *shift*
sorts *mem*, *item*
op $new : mem;\quad write : mem \times item \to mem;\ del : mem \to mem;read : mem \to item$
$\forall\ m : mem;\ e : item;$
• $read(write(new,\ e)) = e$
• $del(write(new,\ e)) = new$
• $@_{lifo} read(write(m,\ e)) = e$
• $@_{fifo} (m{=}new) \Rightarrow read(write(m,\ e)) = e$
• $@_{fifo} \neg\ (m{=}new) \Rightarrow read(write(m,\ e)) = read(m)$
• $@_{lifo} del(write(m,\ e)) = m$
• $@_{fifo} (m{=}new) \Rightarrow del(write(m,\ e)) = new$
• $@_{fifo} \neg\ (m{=}new) \Rightarrow del(write(m,\ e)) = write(del(m),\ e)$
• $@_{lifo} < shift > fifo$
• $@_{fifo} < shift > lifo$

A model M for this second specification is given by a Kripke structure as above but realising, in each state, $M_{s_{lifo}}$ and $M_{s_{fifo}}$ as the classical (initial) models for the stack and queue data types, respectively.

Boilerplates for LTS components specification:	
•*System has modes* <set of Mode>	• Nom := Nom $\uplus$ set of Mode
•<Mode> *is a mode*	• Nom := Nom $\uplus$ {Mode}
•*System has events* <set of Event>	• $\Lambda := \Lambda \uplus$ set of Event
•<Event> *is an event*	• $\Lambda := \Lambda \uplus$ {Event}
• *System's interface is defined by*<InterfaceExp>	• $\Sigma :=$ InterfaceExp
Boilerplates for simple transitions:	
•*System changes from* <Mode1> *to* <Mode2> *through event* <Event>	• $@_{\text{Mode1}}\langle\text{Event}\rangle\text{Mode2}$
• *System may change from* <Mode1> *to* <Mode2> *through event* <Event>	• $@_{\text{Mode1}}[\text{Event}]\text{Mode2}$
• <Event> *changes system to* <Mode>	• $[\text{Event}]Mode$
•*There are no transitions into* <Mode> *through* <Event>	• $\neg\langle\text{Event}\rangle Mode$
Boilerplates for transitions tagged by properties:	
• <Event> *changes modes satisfying* <Prop1> *into modes satisfying* <Prop2>	• Prop1 $\Rightarrow$ [Event]Prop2
• <Event> *changes* <Mode> *to modes satisfying* <Prop>	• Mode $\Rightarrow$ [Event]Prop
• <Event>*changes modes satisfying* <Prop> to mode <Mode>	• Prop $\Rightarrow$ [Event] Mode
Boilerplates for properties:	
• *Property* <Prop> *holds in all modes*	• Prop
• *Property* <Prop> *holds in* <Mode>	• $@_{\text{Mode}}$Prop
• *There is no mode satisfying* <Prop>	• $\neg$ Prop
• *There is at least one mode satisfying* <Prop>	• Ew Prop
• *There is exactly one mode satisfying* <Prop>	• $\forall w, v \in W\ [@_v\text{Prop} \wedge @_w\text{Prop}] \Rightarrow v = w$

4 The specification process

We have seen how to go from a collection of boilerplates to a formal specification in a suitable hybrid logic. The latter not only provides a precise semantics to the requirements gathered, but also paves the way to their *validation*. Actually, a central ingredient for the successful integration of a formal methodology in the industrial practice is the existence of effective tool support.

In order to prototype requirements captured by a collection of boilerplates or to validate their internal consistency, the hybrid specifications are translated into *first-order* logic ($\mathcal{FOL}$), so that the software engineer can take advantage of several provers already available for $\mathcal{FOL}$.

The institution-based framework underlying the hybridisation process, which provides a whole pallet of (hybrid) logics for translating requirements, also of-

fers for free the conceptual machinery for this translation to $\mathcal{FOL}$, whenever it exists. Then, the prover toolset HETS [16], a framework specifically designed to support specifications expressed in different institutions, offers suitable tool support. Using a metaphor of [15], 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 analyzers 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. Note that hybrid logic, namely its propositional variant, has already a number of implementations (see e.g. HTAB [8], HYLOTAB [19] and SPARTACUS [7]). Our approach, however, provides a uniform first order logical framework for analysis and verification supporting the whole methodology. Moreover, to the best of our knowledge, richer versions of hybrid logic do lack effective tool support, which makes our approach by translation the only option available.

We can now explain, step-by-step, the overall methodology for requirements elicitation and validation, as depicted in Fig. 1.

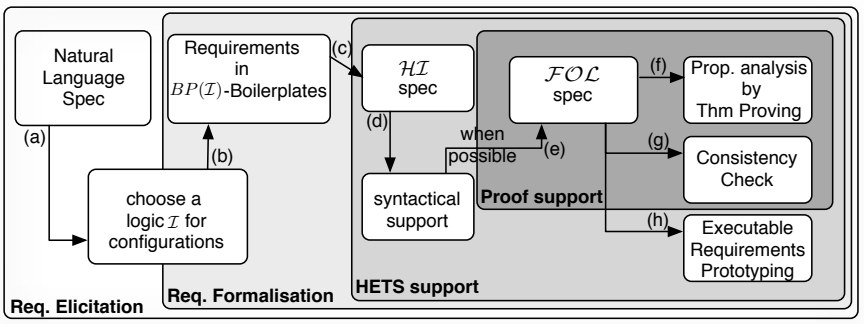

Fig. 1. Tool support

(a),(b) As usual, requirements start from a set of basic facts about what is perceived as the system's goals and constraints. Typically, this determines the choice of a *base* logic $\mathcal{I}$ for expressing properties of local configurations. Examples in propositional and equational logic were discussed above. Often, however, more complex languages are required. One can, for example, specify configurations as *multialgebras* to cope with non determinism, in which

case a multi-valued logic would be the obvious choice. Another possibility to explore is resorts to *partial equational logic* to deal with exceptions, or *observational logics* to specify systems whose configurations encapsulate hidden state-spaces. Finally, if each configuration is itself presented as a transition system, one may choose a *modal logic* as a base, ending up with a (global) modal language to express evolution of modal (local) specifications. This freedom of choosing a base logic for each application is in line with a basic engineering concern which recommends that the choice of a specification framework depends on the nature of the requirements one has to deal with.

Once $\mathcal{I}$ is fixed, the systems requirements are captured in $BP(\mathcal{I})$ instantiation of boilerplates. Note that the set of boilerplates proposed enforces a specification organised in terms of a structured transition system.

(c),(d) The next stage is the translation of the collection of boilerplates $BP(\mathcal{I})$ into a specification in the corresponding hybrid logic $\mathcal{HI}$ according to Boilerplates Table. This specification can be recognized as a HETS specification using the HCASL package recently introduced by the authors in [17].

(e) The existence of a suitable translation, technically a *comorphism* [3], from $\mathcal{HI}$ to $\mathcal{FOL}$ gives, for free, access to a number of provers integrated in HETS in which requirements can be validated. Such a translation, as noticed above, is not available for all logics. References [13,4], however, do provide a roadmap for addressing this issue: [13] shows that the hybridisation of an institution with a comorphism to $\mathcal{FOL}$ also has a comorphism to $\mathcal{FOL}$. Then reference [4] extends this result and characterizes conservativity of those translations to define in which cases it is possible to borrow, in an effective way, proof support from $\mathcal{FOL}$. Note that the proof of this result is is constructive, offering a method to implement such translations. In practice, this is a very general, broadly applicable result since several specification logics do have a comorphism to $\mathcal{FOL}$. Such is the case, for example, of propositional, equational, first-order, modal or even hybrid logic among many others.

Once framed in HETS, the requirement specifications can be validated resorting to several provers for $\mathcal{FOL}$ already "plugged" into HETS [15], e.g., SOFTFOL, SPASS and MATHSERVE BROKER, among others. Additionally, one may also to take advantage of a number of other provers borrowed from other institutions through comorphisms with source in $\mathcal{FOL}$.

(f),(g) Several other features of HETS can be explored in the context of the methodology proposed here. For instance, the model finder of Darwin, which is already integrated in the platform, may be used as a consistency checker for specifications derived from requirements. On the other hand, encodings of $\mathcal{FOL}$ into HASCASL[18], a specification language for functional programs, open new perspectives for prototyping $BP(\mathcal{I})$ generated specifications in a standard programming language as HASKELL.

5 Concluding

The paper proposes a pallet of boilerplates requirements elicitation of reconfigurable systems, as a first step to the definition of a *domain specific language* for this domain of software technology. The pallet is, obviously, not closed, provided that every extension comes equipped with a translation scheme. The combination of different sets of requirements expressed in hybridised versions $\mathcal{HI}$ of different base logics $\mathcal{I}$ is also an interesting strategy to take.

The hybridisation method introduced in [13], which, underlies the construction of suitable specification languages is also able to cope with quantification modalities (i.e., the system's events), a feature which may lead to an enrichment of the boilerplates pallet available at the time of writing. This may provide semantics for boilerplates able to express deadlock situations or to specify more than one-step (ir)-reversibility transition properties. Unfortunately the introduction of nominal quantification rules out the possibility of a suitable first order encoding for the logic, thus reducing the method tool support. Encodings to second-order-logic are, however, being developed.

A known limitation of the method proposed in this paper concerns interface reconfiguration. Technically, service functionality and behaviour exhibited in all system's configurations need to be specified over a common first-order signature. This difficulty was overcome, to a large extent, in a recent publication [12].

Acknowledgements. This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme and by National Funds through FCT, the Portuguese Foundation for Science and Technology, project FCOMP-01- 0124-FEDER-028923.

References

1. P. Blackburn. Arthur Prior and hybrid logic. *Synthese*, 150(3):329–372, 2006.
2. T. Brauner. *Hybrid Logic and its Proof-Theory.* Applied Logic Series. Springer, 2010.
3. R. Diaconescu. *Institution-independent Model Theory.* Studies in Universal Logic. Birkhäuser Basel, 2008.
4. R. Diaconescu and A. Madeira. Encoding hybridized institutions into first order logic. (Submited), 2013.
5. R. Diaconescu and I. Tutu. On the algebra of structured specifications. *Theor. Comput. Sci.*, 412(28):3145–3174, 2011.
6. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *J. ACM*, 39(1):95–146, 1992.
7. D. Götzmann, M. Kaminski, and G. Smolka. Spartacus: A tableau prover for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 262:127–139, 2010.
8. G. Hoffmann and C. Areces. Htab: a terminating tableaux system for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 231:3–19, 2009.
9. M. E. C. Hull, K. Jackson, and J. Dick. *Requirements engineering (2nd ed.).* Springer Verlag, 2005.
10. A. Madeira. *Foundations and techniques for software reconfigurability.* PhD thesis, University of Minho, Portugal (Joint MAP-i Doctoral Program), 2013.

11. A. Madeira, J. M. Faria, M. A. Martins, and L. S. 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.
12. A. Madeira, R. Neves, M. A. Martins, and L. S. Barbosa. When even the interface evolves ... In H. Wang and R. Banach, editors, *Proceedings of TASE (7th IEEE Symp. on Theoretical Aspects of Software Engineering, Birmingham, July, 2003.)*, pages 79–82. IEEE Computer Society, 2013.
13. M. A. Martins, A. Madeira, R. Diaconescu, and L. S. 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.
14. T. Mossakowski. Foundations of heterogeneous specification. In M. Wirsing, D.Pattinson, and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques (Revised Selected Papers of WADT 2002, Frauenchiemsee, Germany, September 24-27, 2002)*, volume 2755 of *Lecture Notes in Computer Science*, pages 359–375. Springer, 2003.
15. T. Mossakowski, C. Maeder, M. Codescu, and D. Lucke. HETS User Guide - Version 0.99. Technical report, DFKI Lab Bremen, April 2013.
16. T. Mossakowski, C. Maeder, and K. 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.
17. R. Neves, A. Madeira, M. A. Martins, and L. S. Barbosa. Hybridisation at work. In *CALCO TOOLS*, volume (to appear) of *Lecture Notes in Computer Science*. Springer, 2013.
18. L. Schröder and T. Mossakowski. Hascasl: Towards integrated specification and development of functional programs. In H. Kirchner and C. Ringeissen, editors, *AMAST*, volume 2422 of *Lecture Notes in Computer Science*, pages 99–116. Springer, 2002.
19. J. van Eijck. Hylotab-tableau-based theorem proving for hybrid logics. Technical report, CWI, Amsterdam, 2002.

Appendix: the hybridisation process

This appendix provides a brief overview of the hybridisation method which allows for the systematic construction of hybrid languages from arbitrary logics. The method is framed in the theory of institutions whose basic definitions are recalled.

Institutions

An *institution* is a category theoretic formalisation of a logical system, encompassing syntax, semantics and satisfaction. The concept was put forward by Goguen and Burstall, in the end of the seventies, in order to *"formalise the formal notion of logical systems"*, in response to the *"population explosion among the logical systems used in Computing Science"* [6]. Formally,

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

- a category $\mathrm{Sign}^{\mathcal{I}}$ of *signatures* and *signature* morphisms,
- a functor $\mathrm{Sen}^{\mathcal{I}} : \mathrm{Sign}^{\mathcal{I}} \to \mathbb{S}et$ giving for each signature a set whose elements are called *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 called *Σ-models*, and whose arrows are called *Σ-(model) homomorphisms*, and
- 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

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

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

Example 1 (Propositional Logic).

A signature $Prop \in |\mathrm{Sign}^{PL}|$ is a set of propositional variables symbols and a signature morphism is just a function $\varphi : Prop \to Prop'$ Therefore, Sign^{PL} coincides with the category $\mathbb{S}et$.

Functor Mod maps each signature $Prop$ to the category $\mathrm{Mod}^{PL}(Prop)$ and each signature morphism φ to the reduct functor $\mathrm{Mod}^{PL}(\varphi)$. Objects of $\mathrm{Mod}^{PL}(Prop)$ are functions $M : Prop \to \{\top, \bot\}$ and, its morphisms, functions $h : Prop \to Prop$ such that $M(p) = M'(h(p))$. Given a signature morphism $\varphi : Prop \to Prop'$, the reduct of a model $M' \in |\mathrm{Mod}^{PL}(Prop')|$, say $M = \mathrm{Mod}^{PL}(\varphi)(M')$ is defined, for each $p \in Prop$, as $M(p) = M'(\varphi(p))$.

The sentences functor maps each signature $Prop$ to the set of propositional sentences $\mathrm{Sen}^{PL}(Prop)$ and each morphism $\varphi : Prop \to Prop'$ to the sentences' translation $\mathrm{Sen}^{PL}(\varphi) : \mathrm{Sen}^{PL}(Prop) \to \mathrm{Sen}^{PL}(Prop')$. The set $\mathrm{Sen}^{PL}(Prop)$ is the usual set of propositional formulae defined by the grammar

$$\rho ::= p \mid \rho \vee \rho \mid \rho \wedge \rho \mid \rho \Rightarrow \rho \mid \neg\rho$$

for $p \in Prop$. The translation of a sentence $\mathrm{Sen}^{PL}(\varphi)(\rho)$ is obtained by replacing each proposition of ρ by the respective φ-image. Finally, for each $Prop \in \mathrm{Sen}^{PL}$, the satisfaction relation $\models^{PL}_{Prop}$ is defined as usual:

- $M \models^{PL}_{Prop} p$ iff $M(p) = \top$, for any $p \in Prop$;
- $M \models^{PL}_{Prop} \rho \vee \rho'$ iff $M \models^{PL}_{Prop} \rho$ or $M \models^{PL}_{Prop} \rho'$,

and similarly for the other connectives.

Example 2 (Equational logic).

Signatures in the institution EQ of equational logic are pairs (S, F) where S is a set of sort symbols and $F = \{F_{\underline{ar} \to s} \mid \underline{ar} \in S^*, s \in S\}$ is a family of sets of operation symbols indexed by arities $\underline{ar}$ (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 $\varphi^{\mathrm{op}} = \{\varphi^{\mathrm{op}}_{\underline{\mathrm{ar}}\to s} :\ F_{\underline{\mathrm{ar}}\to s} \to F'_{\varphi^{\mathrm{st}}(\underline{\mathrm{ar}})\to\varphi^{\mathrm{st}}(s)} \mid \underline{\mathrm{ar}} \in S^*, s \in S\}$ 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 morphism 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 universal 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 first-order 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.

The hybridisation method

Having recalled the notion of an institution, we shall now briefly review the core of the *hybridisation* method proposed in [13,4]. For the sake of brevity, we shall restrict ourselves to a simplified (quantifier-free and non-constrained) version of the general method.

As explained in the paper, the method enriches a base (arbitrary) institution $\mathcal{I} = (\mathrm{Sign}^{\mathcal{I}}, \mathrm{Sen}^{\mathcal{I}}, \mathrm{Mod}^{\mathcal{I}}, (\models^{\mathcal{I}}_{\Sigma})_{\Sigma\in|\mathrm{Sign}^{\mathcal{I}}|})$ with hybrid logic features and the corresponding Kripke semantics. The result is still an institution, $\mathcal{HI}$, called the *hybridisation of* $\mathcal{I}$.

The category of $\mathcal{HI}$-signatures. The base signature is 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:

$$\mathrm{Sign}^{\mathcal{HI}} = \mathrm{Sign}^{\mathcal{I}} \times \mathrm{Sign}^{REL}.$$

Thus, signatures are triples $(\Sigma, \mathrm{Nom}, \Lambda)$, where $\Sigma \in |\mathrm{Sign}^{\mathcal{I}}|$ and, in the *REL*-signature (Nom, Λ), Nom is a set of constants called *nominals* and Λ is a set of relational symbols called *modalities*; Λ_n stands for the set of modalities of arity n. Morphisms $\varphi \in \mathrm{Sign}^{\mathcal{HI}}((\Sigma, \mathrm{Nom}, \Lambda), (\Sigma', \mathrm{Nom}', \Lambda'))$ are triples $\varphi = (\varphi_{\mathrm{Sig}}, \varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}})$ where $\varphi_{\mathrm{Sig}} \in \mathrm{Sign}^{\mathcal{I}}(\Sigma, \Sigma')$, $\varphi_{\mathrm{Nom}} : \mathrm{Nom} \to \mathrm{Nom}'$ is a function

and $\varphi_{\mathrm{MS}} = (\varphi_n : \Lambda_n \to \Lambda'_n)_{n\in\mathbb{N}}$ a $\mathbb{N}$-family of functions mapping nominals and $n-ary$-modality symbols, respectively.

$\mathcal{HI}$-sentences functor. The second step is to enrich the base sentences accordingly. The sentences of the base institution and the nominals are taken as atoms and composed with the boolean connectives, modalities, and satisfaction operators as follows: $\mathrm{Sen}^{\mathcal{HI}}(\Sigma, \mathrm{Nom}, \Lambda)$ is the least set such that

- $\mathrm{Nom} \subseteq \mathrm{Sen}^{\mathcal{HI}}(\Delta)$;
- $\mathrm{Sen}^{\mathcal{I}}(\Sigma) \subseteq \mathrm{Sen}^{\mathcal{HI}}(\Delta)$;
- $\rho \star \rho' \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$ for any $\rho, \rho' \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$ and any $\star \in \{\vee, \wedge, \Rightarrow\}$,
- $\neg\rho \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$, for any $\rho \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$,
- $@_i\rho \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$ for any $\rho \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$ and $i \in \mathrm{Nom}$;
- $[\lambda](\rho_1, \ldots, \rho_n), \langle\lambda\rangle(\rho_1, \ldots, \rho_n) \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$, for any $\lambda \in \Lambda_{n+1}, \rho_i \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$, $i \in \{1, \ldots, n\}$.

Given a morphism $\varphi = (\varphi_{\mathrm{Sig}}, \varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}}) : (\Sigma, \mathrm{Nom}, \Lambda) \to (\Sigma', \mathrm{Nom}', \Lambda')$, the translation of sentences $\mathrm{Sen}^{\mathcal{HI}}(\varphi)$ is defined as follows:

- $\mathrm{Sen}^{\mathcal{HI}}(\varphi)(\rho) = \mathrm{Sen}^{\mathcal{I}}(\varphi_{\mathrm{Sig}})(\rho)$ for any $\rho \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$;
- $\mathrm{Sen}^{\mathcal{HI}}(\varphi)(i) = \varphi_{\mathrm{Nom}}(i)$;
- $\mathrm{Sen}^{\mathcal{HI}}(\varphi)(\neg\rho) = \neg\mathrm{Sen}^{\mathcal{HI}}(\varphi)(\rho)$;
- $\mathrm{Sen}^{\mathcal{HI}}(\varphi)(\rho \star \rho') = \mathrm{Sen}^{\mathcal{HI}}(\varphi)(\rho) \star \mathrm{Sen}^{\mathcal{HI}}(\varphi)(\rho')$, $\star \in \{\vee, \wedge, \Rightarrow\}$;
- $\mathrm{Sen}^{\mathcal{HI}}(\varphi)(@_i\rho) = @_{\varphi_{\mathrm{Nom}}(i)}\mathrm{Sen}^{\mathcal{HI}}(\rho)$;
- $\mathrm{Sen}^{\mathcal{HI}}(\varphi)([\lambda](\rho_1, \ldots, \rho_n)) = [\varphi_{\mathrm{MS}}(\lambda)](\mathrm{Sen}^{\mathcal{HI}}(\rho_1), \ldots, \mathrm{Sen}^{\mathcal{HI}}(\rho_n))$;
- $\mathrm{Sen}^{\mathcal{HI}}(\varphi)(\langle\lambda\rangle(\rho_1, \ldots, \rho_n)) = \langle\varphi_{\mathrm{MS}}(\lambda)\rangle(\mathrm{Sen}^{\mathcal{HI}}(\rho_1), \ldots, \mathrm{Sen}^{\mathcal{HI}}(\rho_n))$.

$\mathcal{HI}$-models functor. Models of the hybridised logic $\mathcal{HI}$ can be regarded as (Λ-)Kripke structures whose worlds are $\mathcal{I}$-models. Formally $(\Sigma, \mathrm{Nom}, \Lambda)$-*models* are pairs (M, W) where

- W is a (Nom, Λ)-model in $\underline{REL}$;
- M is a function $|W| \to |\mathrm{Mod}^{\mathcal{I}}(\Sigma)|$.

In each world (M, W), $\{W_n \mid n \in \mathrm{Nom}\}$ provides interpretations for *nominals* in Nom, whereas relations $\{W_\lambda \mid \lambda \in \Lambda_n, n \in \omega\}$ interprete *modalities* in Λ. We denote $M(w)$ simply by M_w. The reduct definition is lifted from the base institution: the reduct of a Δ'-model (M', W') along a signature morphism $\varphi = (\varphi_{\mathrm{Sig}}, \varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}}) : \Delta \to \Delta'$, denoted by $\mathrm{Mod}^{\mathcal{HI}}(\varphi)(M', W')$, is the Δ-model (M, W) such that

- W is the $(\varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}})$-reduct of W'; i.e.
  - $|W| = |W'|$;
  - for any $n \in \mathrm{Nom}, W_n = W'_{\varphi_{\mathrm{Nom}}(n)}$;
  - for any $\lambda \in \Lambda$, $W_\lambda = W'_{\varphi_{\mathrm{MS}}(\lambda)}$;

  and
- for any $w \in |W|$, $M_w = \mathrm{Mod}^{\mathcal{I}}(\varphi_{\mathrm{Sig}})(M'_w)$.

Satisfaction. Let $(\Sigma, \mathrm{Nom}, \Lambda) \in |\mathrm{Sign}^{\mathcal{HI}}|$ and $(M, W) \in |\mathrm{Mod}^{\mathcal{HI}}(\Sigma, \mathrm{Nom}, \Lambda)|$. For any $w \in |W|$ we define:

- $(M,W) \models^w \rho$ iff $M_w \models^{\mathcal{I}} \rho$; when $\rho \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$,
- $(M,W) \models^w i$ iff $W_i = w$; when $i \in \mathrm{Nom}$,
- $(M,W) \models^w \rho \vee \rho'$ iff $(M,W) \models^w \rho$ or $(M,W) \models^w \rho'$,
- $(M,W) \models^w \rho \wedge \rho'$ iff $(M,W) \models^w \rho$ and $(M,W) \models^w \rho'$,
- $(M,W) \models^w \rho \Rightarrow \rho'$ iff $(M,W) \models^w \rho$ implies that $(M,W) \models^w \rho'$,
- $(M,W) \models^w \neg\rho$ iff $(M,W) \not\models^w \rho$,
- $(M,W) \models^w [\lambda](\xi_1,\ldots,\xi_n)$ iff for any $(w,w_1,\ldots,w_n) \in W_\lambda$ we have that $(M,W) \models^{w_i} \xi_i$ for some $1 \leq i \leq n$.
- $(M,W) \models^w \langle\lambda\rangle(\xi_1,\ldots,\xi_n)$ iff there exists $(w,w_1,\ldots,w_n) \in W_\lambda$ such that and $(M,W) \models^{w_i} \xi_i$ for any $1 \leq i \leq n$.
- $(M,W) \models^w @_j\rho$ iff $(M,W) \models^{W_j} \rho$,

We write $(M,W) \models \rho$ iff $(M,W) \models^w \rho$ for any $w \in |W|$.

Theorem 1 ([13]). *Let $\Delta = (\Sigma, \mathrm{Nom}, \Lambda)$ and $\Delta' = (\Sigma', \mathrm{Nom}', \Lambda')$ be two $\mathcal{HI}$-signatures and $\varphi : \Delta \to \Delta'$ a morphism of signatures. For any $\rho \in \mathrm{Sen}^{\mathcal{HI}}(\Delta)$, $(M',W') \in |\mathrm{Mod}^C(\Delta')|$, and $w \in |W|$,*

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

The method can be illustrated through its application to the two institutions described above and used in the paper: those of propositional and equational logics.

Example 3 ($\mathcal{H}PL$). The hybridisation of the propositional logic institution PL is an institution where signatures are triples $(Prop, \mathrm{Nom}, \Lambda)$ and sentences are generated by

$$\rho ::= \rho_0 \mid i \mid @_i\rho \mid \rho \odot \rho \mid \neg\rho \mid \langle\lambda\rangle(\rho,\ldots,\rho) \mid [\lambda](\rho,\ldots,\rho) \tag{2}$$

where $\rho_0 \in \mathrm{Sen}^{PL}(Prop)$, $i \in \mathrm{Nom}$, $\lambda \in \Lambda_n$ and $\odot = \{\vee, \wedge, \Rightarrow\}$. Note there is a double level of connectives in the sentences: the one coming from base PL-sentences and another introduced by the hybridisation process. However, they "semantically collapse" and, hence, no distinction between them needs to be done (see [4] for details). A $(Prop, \mathrm{Nom}, \Lambda)$-model is a pair (M,W), where W is a transition structure with a set of worlds $|W|$. Constants $W_i, i \in \mathrm{Nom}$ stand for the named worlds and $(n+1)$-ary relations W_λ, $\lambda \in \Lambda_n$ are the accessibility relations characterising the structure. For each world $w \in |W|$, $M(w)$ is a (local) PL-model, assigning propositions in $Prop$ to the world w.

Restricting the signatures to those with just a single unary modality (i.e., where $\Lambda_1 = \{\lambda\}$ and $\Lambda_n = \emptyset$ for the remaining $n \neq 1$), results in the usual institution for classical hybrid propositional logic [2].

Example 4 ($\mathcal{H}EQ$). Signatures of $\mathcal{H}EQ$ are triples $((S,F), \mathrm{Nom}, \Lambda)$ and the sentences 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 state. Distinct configurations are therefore modeled by distinct algebras and reconfigurations expressed by transitions over a graph of algebras (cf., [11,10]).