On requirements engineering for reactive systems: A formal methodology

Alexandre Madeira[1,2,3], José M. Faria[3,2] Manuel A. Martins[1], and Luís S. Barbosa[2]
[1] Dep. of Mathematics of U. Aveiro; [2]Dep. of Informatics & CCTC of U. Minho
[3] Critical Software S.A, Portugal

Abstract

This paper introduces a rigorous methodology for requirements specification of systems that react to external stimulus and consequently evolve through different operational modes, providing, in each of them, different functionalities. The proposed methodology proceeds in three stages, enriching a simple state-machine with local algebraic specifications. It resorts to an expressive variant of hybrid logic which is latter translated into first-order to allow for ample automatic tool support.

1. Introduction

Motivation and objectives. The successful development and deployment of critical embedded systems, from the early concept and system definition phases, down to implementation and validation, poses a number of challenges that engineers must overcome. In the software development domain, experience in industry has for long shown (see, e.g., [Ber02]) that the most significant problems are due to poor quality of requirements.

Proper management of requirements throughout the complete development life-cycle requires their (i) elicitation, (ii) specification, (iii) verification against ambiguities, omissions, and incoherences, (iv) demonstration of traceability along the different life-cycle stages, (v) adequate management of possible changes imposed by external factors, and (vi) validation against the implementation.

In practice, the best start to overcome many of the difficulties imposed by each of these steps is to adopt good engineering practices, establishing, for example, rigorous processes and well defined borders between, e.g., user, system, hardware, software, functional, and non-functional requirements.

A significant leap beyond engineering maturity can yet be achieved by adopting advanced techniques that provide rigorous and machinery support for all these tasks, supported by well founded mathematical semantics.

There are two basic frameworks to formally capture software requirements: one emphasizes *behaviour* and its evolution; the other focus *data* and their transformations.

Reactive systems are typically specified through (some variant of) *state-machines*. Such models capture system's evolution in terms of event occurrence and its impact in the system internal state configuration. Automata theory, and its more recent, abstract rendering in coalgebraic terms, provide a suitable formalism for both specification and analysis. Crucial notions of bisimulation, minimization and invariant, among others, play a fundamental, long established in this framework.

The dual specification paradigm is focused on system's functional behavior given in terms of input-output relations modeling operations on *data*. It is, therefore, essentially algebraic: a specification is a theory in a suitable logic, expressed over a signature, which captures its syntactic interface. Specification models consists of concrete algebras which satisfy the specified theory [DF02, MHST03].

In practice, however, the aspects considered on both approaches are interconnected: the functionality offered by the system system, at each moment, may depend on the stage of its evolution. Such is typically the case of complex, reconfigurable software.

This paper explores such a interconnection. Starting from a classical state-machine specification, the approach illustrated in the sequel, goes a step further: different states are inter-

preted as different *modes* of operation and each of them is equipped with an algebraic specification (over the system's interface) of the corresponding functionality. Technically, specifications become *structured* state-machines, states denoting *algebras*, rather than *sets*. We call them *states-as-algebras* models. The remaining of this paper introduces their specification and illustrates their use through a (partial) view of a cruise control system [HKL97].

Outline. Section 2 introduces the proposed methodology and the application example. A classical, state-machine specification, resorting to a form of hybrid logic [Bla00] is provided in section 3. Section 4 introduces an alternative *states-as-algebras* model for the same problem. Finally, Section 5 shows that specification in the hybrid logic used can be translated to first order logic (FOL), paving the way to suitable tool support. Section 6 concludes, evaluating the extent to which our methodology covers the above identified (i-vi) tasks.

2. A specification methodology

Figure 1 sums up the proposed methodology. The block on the left hand side represents the specification framework, structured in two stages, as explained below. The annotation on top — *Hybrid logic* — states the underlying logic used. The block on the right concerns verification and analysis of specifications suitably translated to FOL. The translation itself is depicted as a *comorphism* between the two logic systems in presence: hybrid logic, chosen for its expressive power, first order for existent verification support. Before detailing the methodology, let us explain both the choice of hybrid logic and the translation process.

Modal and temporal logics are widely used to express properties of reactive systems. However, they lack any mechanism to make explicit references to specific states in a model. Such limitation is overcome with hybrid logic[1] [Bla00], through the introduction of special symbols, called *nominals* to reference individual states This is achieved through a family of connectives $@_i$, indexed by nominals i: intuitively $@_i\, p$ states the validity of p at the state identified by nominal i.

1 Observe that qualifier *hybrid* in hybrid logic has nothing to do with the usual meaning the word has in computer science, e.g. in 'hybrid systems'.

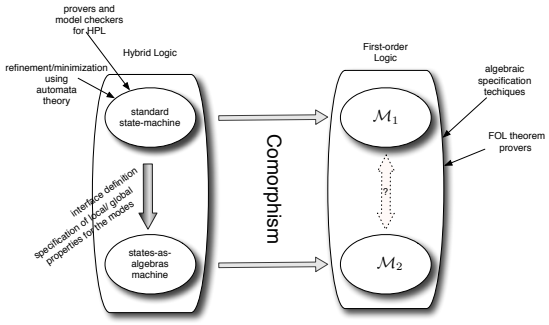

Figure 1. Specification methodology

A basic property to require from a specification formalism is its ability to be framed as an *institution* [GB92, Dia08]. This is not a formal idiosyncracy: institutions, as abstract, general representations of logical systems, provide modular structuring and parameterization mechanisms which are defined 'once and for all' abstracting from the concrete particularities of the intended logics [Tar03], with suitable tool support. CASL [MHST03], for example, is a well known system where the semantics of a set of combinators for specifications are characterized at such a level of generality. Moreover institutions provide a systematic way to relate logics and transport results from one to another [Mos03], which means that a theorem prover for the latter can be used to reason about specifications written in the former. This is achieved through a special call of maps between institutions, referred to as *comorphism*, as depicted in Figure 1.

The methodology proposed in this paper is, therefore, institution-based. That our hybrid formalism constitutes an institution is proved in [MMDB11]. In the same reference a comorphism from this institution into FOL is established, which allows to translate any specification expressed in our logic into an equivalent first order specifications. The real relevance of this result cannot be understated: it is well-known that FOL enjoys mature tool support which becomes available to verification of our own specifications. In particular, in this paper, the HETS-*heterogeneous tool set* [MML07] was used to analyze the examples reported here, once translated to FOL. HETS provides a proof management tool which incorporates a number of theorem provers.

Let us now detail the proposed methodology, as

depicted in Figure 1, in three main stages:

I (I.1) Express the requirements in *hybrid propositinal logic* (HPL), identifying states and transitions to build a first state-machine; (I.2) Specify local properties as propositions; At this stage, traditional technics of state machine analysis/refinement may be applied, and available reasoning tools for HPL used (see Section 3).

II (II.1)Define the actual system's interface through the set of (external) services offered. Technically, this is supported by the definition of a (multi-sorted) algebraic signature. (II.2) Express, whenever possible, the attributes of the first machine as functional properties over this signature.

III Translate both specifications into FOL, providing a common ground for testing and verification.

In the sequel the methodology is illustrated in a number of specification fragments of a *automatic cruise control* (ACC) system. The example is taken from [HKL97]:

> *"The mode class CruiseControl contains four modes, Off, Inactive, Cruise, and Override. At any given time, the system must be in one of these modes. Turning the ignition on causes the system to leave Off mode and enter Inactive mode, while turning the cruise control level to const when the brake is off and the engine running causes the system to enter Cruise mode. (...) Once cruise control has been invoked, the system uses the automobile's actual speed to determine whether to set the throttle to accelerate or decelerate the automobile, or to maintain the current speed (...)To override cruise control (i.e., enter Override), the driver turns the lever to off or applies the brake".*

3. Hybrid specifications

The requirements for the cruise control system example can be modeled as the state machine depicted in Figure 2. This section introduces its specification in propositional hybrid logic (HPL). Such a presentation has the advantage of being compact, unambiguous and closer to the input format of typical verification engines.

The set of HPL formulas is defined by the following grammar:

$$\varphi, \psi ::= p \mid i \mid \neg\varphi \mid [\lambda]\varphi \mid @_i\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \qquad (1)$$

where λ ranges over a set Λ of modal operators. Models of this logic are state-machines with an additional function $state : Nom \to S$ which assigns each nominal to the respective state. Thus, models are tuples $\mathcal{P} = \langle S, state, (R_\lambda)_{\lambda\in\Lambda}, (P_s)_{s\in S}\rangle$ where S is a set of states, $R_\lambda \subseteq S \times S$ is the accessibility relation associated to the modality λ and $P_s : Prop \to \{\top, \bot\}$ is the function that assigns the propositions on the state $s \in S$. The satisfaction relation is defined as on the standard modal logic (e.g.

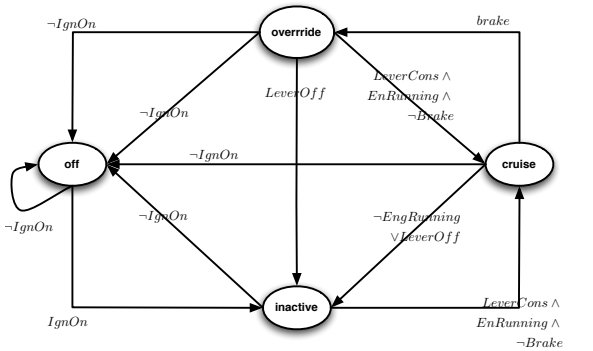

Figure 2. State-machine of the system

$\mathcal{P} \models^s p$ iff $P_s(p) = \top$; $\mathcal{P} \models^s [\lambda]\varphi$ iff $\mathcal{P} \models^{s'} \varphi$ for any s' such that $(s, s') \in R_\lambda$) adding the cases

- $\mathcal{P} \models^s @_i\varphi$ iff $\mathcal{P} \models^{state(i)} \varphi$;
- $\mathcal{P} \models^s i$ iff $state(i) = s$.

Moreover, we abbreviate formulas $\neg[\lambda]\neg\varphi$ with $\langle\lambda\rangle$ and formulas $\langle\lambda\rangle\varphi \wedge [\lambda]\varphi$ with $\langle\lambda\rangle^\circ\varphi$. For our running example, a modality $\{next\}$ is introduced to denote the state-machine accessibility relation. Nominals in $\{off, inactive, override, cruise\}$ correspond to the above mentioned modes. Finally, a set of propositions is considered corresponding to labels in Figure 2. With such signature, transitions are specified as follows:

- $(T_1)@_{off}(\ IgnOn \Rightarrow \langle next\rangle^\circ inactive\ )$
- $(T_2)\neg\ IgnOn \Rightarrow \langle next\rangle^\circ off$
- $(T_3)@_{inactive}(LeverCons \wedge IgnOn \wedge \neg\ Brake \Rightarrow \langle next\rangle^\circ cruise)$
- $(T_4)@_{cruise}(\neg\ EngRunning \vee LeverOff \Rightarrow \langle next\rangle^\circ inactive)$
- $(T_5)@_{cruise}(\ Brake \Rightarrow \langle next\rangle^\circ override)$
- $(T_6)@_{override}(LeverCons \wedge IgnOn) \wedge EngRunning \wedge \neg\ Brake \Rightarrow \langle next\rangle^\circ cruise)$

We are also able to express local properties, individual states being referenced through the satisfaction operator $@_i$, where i is the corresponding nominal. For instance, the requirement that the engine controls speed decelerating the car if the *speed is high* and maintaining it when it is considered *adequate* is modelled by

- $(L^1_{cruise})@_{cruise}(\ IgnOn \wedge EngRunning \wedge HighSpeed \Rightarrow decel)$
- $(L^2_{cruise})@_{cruise}(IgnOn \wedge EngRunning \wedge AdmissibleSpeed \Rightarrow mantain)$

Admissibility properties, concerning propositions, are also captured. For instance, the fact that *the lever cannot be switched in more than one position at each time*, and equally so for the acceleration and speed modes, is expressed as

- $(A_1)LeverOff \Leftrightarrow \neg\ LeverCons$
- $(A_3)accel \Rightarrow \neg\ decel \wedge \neg\ mantain$
- ...
- $(A_4)HighSpeed \Rightarrow \neg\ CruiseSpeed \wedge \neg\ LowSpeed$

As it stands, this hybrid specification enjoys a rich tool support through recent implementations of logical calculus for HPL (e.g. HTab [HA09], HyLoTab [vE02] and Spartacus [GKS10]). Moreover, model checking for HPL models is also an active research issue (e.g. [Lan09, HS08]). Our focus is, however, a different, somehow more santdard, one: hybrid specifications are translated to FOL through a suitable comorphism. This solution 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, as required for the 'states-as-algebras specification' stage described in the following section, lack effective tool support, which makes our approach by translation the only option available.

4. States as algebras

The logic. The second stage in the envisaged methodology equips each state in the underlying state-machine with an *algebra* modeling its local functionality. Therefore, models are enriched with a family of algebras indexed by the set of states, i.e., they become structures $\mathcal{M} = \langle S, state, (R_\lambda)_{\lambda\in\Lambda}, (P_s)_{s\in S}, (A_s)_{s\in S}\rangle$, where algebras[2] in the family $(A_s)_{s\in S}$ are defined over the same signature and universe. Each A_s models the system behaviour on state $s \in S$. Taking the algebraic signature Σ (of the system interface), sentences correspond to the set of formulas defined by the following grammar:

$$\varphi, \psi := p \mid i \mid e \mid P(\bar{t}) \mid \neg\varphi \mid \varphi \star \psi \mid [\lambda]\varphi \mid @_i\varphi \mid \forall x\varphi \qquad (2)$$

where $\star \in \{\vee, \wedge, \Rightarrow\}$, p is a proposition, i is a nominal, e is a Σ-equation over a contable infinite sorted-set of variables X, P is a Σ-predicate of type $s_1, \ldots, s_n$ where $\bar{t} := t_1, \ldots, t_n$ and $t_i \in (T_\Sigma)_{s_i}$. An assignment for $\mathcal{M}$ consists of a function $g : X \to A$, where A is the carrier set of the algebras of $\mathcal{M}$. Satisfaction relation on these structures is defined as follows:

- $\mathcal{M}, g \models^s i$ if $state(i) = s$;
- $\mathcal{M}, g \models^s p$ if $P_s(p) = \top$;
- $\mathcal{M}, g \models^s e$ if $A_s \models e[g]$;
- $\mathcal{M}, g \models^s P(t_1, \ldots, t_n)$ if $A_s \models P(t_1, \ldots, t_n)[g]$;
- $\mathcal{M}, g \models^s \varphi \vee \varphi'$ if $\mathcal{M}, g \models^s \varphi$ or $\mathcal{M}, g \models^s \varphi'$;
- $\mathcal{M}, g \models^s \forall x\varphi$ if, for any assignment g' such that for any $y \neq x$, $g(y) = g'(y)$, one has $\mathcal{M}, g' \models^s \varphi$.

2 Strictly speaking they are first-order-structures, the language abuse is justified by the fact that, in a multi-sort specification, predicates can be regarded as boolean-valued functions.

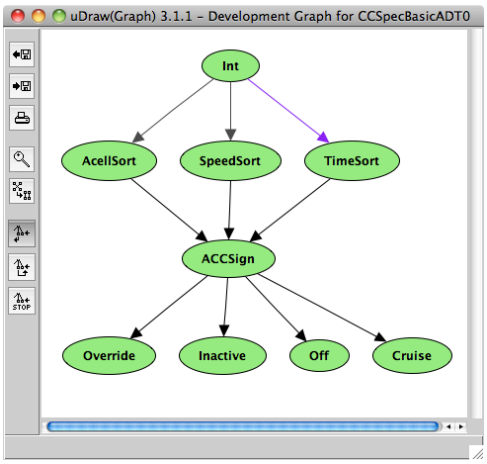

Figure 3. HETS development graph

- $\mathcal{M}, g \models^s [\lambda]\varphi$ if, for any $s' \in S$ such that $(s, s') \in R_\lambda$, one has $\mathcal{M}, g \models^{s'} \varphi$.

and similarly for the remaining boolean connectives. We write $\mathcal{M} \models^s \varphi$ when for any assignment g, $\mathcal{M}, g \models^s \varphi$.

The interface. In order to model the system's functionality as physically provided by the car artifact, we resort to a classical algebraic specification. This entails the need for introducing data types able to support the envisaged notions of *time*, *speed* and *acceleration*. In a simple model integer numbers, with the usual operations and predicates $\{+, \leq, \geq, <, >\}$, can do the job.

spec TimeSort =
  Int **with sort** $Int \mapsto time$,
ops $0 \mapsto init$, $suc \mapsto after$ **end**
spec SpeedSort =
  Int **with sort** $Int \mapsto speed$ **end**
spec AcellSort =
  Int **with sort** $Int \mapsto accel$ **end**

There is also a need to consider a set of operation symbols to represent the interface of the system (technically, a signature). Thus, the operation *Pedal* models the accelerations applied by the driver at each moment. On the other hand, *Automatic* captures accelerations applied on the engine by the ACC, and *CurrentSpeed* records the current speed. Finally, constant *MaxCruiseSpeed* represents the maximum speed allowed on the ACC mode:

spec ACCSign =
  TimeSort **and** SpeedSort **and** AcellSort
then ops $Pedal : time \to accel$;
  $Automatic : time \to accel$;
  $Speed : speed \times accel \to speed$;
  $CurrentSpeed : time \to speed$;
  $MaxCruiseSpeed : speed$

There are properties that globally hold, in all the configurations of the system. For instance,

$\forall\ s : speed; a : accel$
- $(G_1)\ Speed(s, a) \geq 0$
- $(G_2)\ CurrentSpeed(t) = 0 \wedge Pedal(t) \geq 0$
  $\Rightarrow CurrentSpeed(after(t)) \geq 0$
- $(G_3)\ Pedal(t) \geq 0 \Leftrightarrow$
  $CurrentSpeed(t) < CurrentSpeed(after(t))$
- $(G_4)\ Speed(s, a) = s \Leftrightarrow a = 0$
- $(G_5)\ CurrentSpeed(after(t)) =$
  $Speed(CurrentSpeed(t), Pedal(t))$

Local properties. Differently from the properties above, local requirements hold only on particular configurations. Let us explore some of them. First, in state off, it is required that speed and acceleration are null and no other operations in the interface react:

$\forall\ t : time; s : speed; a : accel$
- $(L^1_{off})\ @_{off}\ CurrentSpeed(t) = 0$
- $(L^2_{off})\ @_{off}\ Speed(s, a) = 0$

In state *inactive*, the speed and acceleration depend on the accelerations automatically introduced in the system, i.e,

$\forall\ s : speed; a : accel$
- $(L^1_{inactive})\ @_{inactive} Speed(s, a) = s + a$

Whenever the cruise control is active, control gets less simple. Thus, if the car's speed is higher than $MaxCruiseSpeed$ it has to decelerate until reaching $MaxCruiseSpeed$. If the speed is lower that $MaxCruiseSpeed$, the car should pursue at the current speed. To capture these requirements, (L^1_{cruise}) and (L^2_{cruise}) are replaced by

$\forall\ t: time; s : speed; a : accel$
- $(L^{1'}_{cruise})\ @_{cruise}[CurrentSpeed(t) > MaxCruiseSpeed \Rightarrow$
  $Automatic(after(t)) < 0]$
- $(L^{2'}_{cruise})\ @_{cruise}[CurrentSpeed(t) \leq MaxCruiseSpeed$
  $\Leftrightarrow Automatic(after(t)) = 0]$
- $(L^3_{cruise}) @_{cruise} Speed(s, a) = s + a$
- $(L^4_{cruise}) @_{cruise} Pedal(t) \geq 0 \Rightarrow Pedal(t) = Automatic(t)$

An interesting feature in this example is that properties local to states *override* and off do coincide. The system's behaviour on both states only differs in what concerns the definition of the allowed transitions. The latter are dealt as follows.

Transitions specification. To specify state transitions we simply resort to the state-machine built in the first stage of our methodology, through axioms $(T_1), \ldots (T_n)$ from Section 3. However, some propositions may now be expressed by means of algebraic properties of local states. For instance, we may replace (T_4) by

$\forall\ t$: $time$
- $(T_{4'}) @_{cruise}[CurrentSpeed(t) = 0 \vee LeverOff$
  $\Rightarrow \langle next \rangle^{\circ}(inactive \wedge CurrentSpeed(after(t)) = 0)].$

Furthermore, the fact that when ACC is activated by transition T_6, the speed should to be maintained, is captured by

$\forall\ t$: $time$; $\forall\ s$: $speed$
- $(T_{6'}) @_{override}[(LeverCons \wedge CurrentSpeed(t) = s\ \wedge\ s \geq 0)$
  $\Rightarrow \langle next \rangle^{\circ}(cruise \wedge CurrentSpeed(after(t)) = s)].$

5. Translating to FOL

This section briefly illustrates how hybrid-specifications are translated to FOL. For lack of space the formal definition of the comorphism which actually does the job is omitted here (details and proof are available in report [MMDB11] to which the interested reader is referred). The translation proceeds through the inclusion of a special sort $\overline{st}$ which denotes states. Hence, in order to 'collapse' all the local state algebras in a unique structure, the signature of all operations and predicates is enriched with an argument of this sort. For instance, predicate $\leq$: $speed \times speed$ is transformed into $\leq^*$: $st^* \times speed \times speed$. On the other hand, nominals are regarded as constants over $\overline{st}$, modalities as usual first-order relations and propositions as unary predicates over $\overline{st}$. Therefore, we end up with the following signature

ops
$Speed^* : st^* \times speed \times accel \rightarrow speed;$
$Pedal^* : st^* \times time \rightarrow accel; \ldots$
pred
$next : st^* \times st^*; IgnOn^* : st^*; \ldots$

Note that, now, global properties are universally quantified, and local properties take as state argument the respective nominal. For instance, global properties (G_1) and (G_2) are translated into

$\forall\ s : speed; w : st^*; a : accel; t : time$
- $(G_{1^*})\ \geq^*(w, Speed^*(w, s, a), 0^*(w))$
- $(G_{2^*})\ CurrentSpeed^*(w,t) = 0^*(w) \wedge \geq^*(w, Pedal^*(w,t), 0^*(w)).$

and local properties (L^1_{off}) and (L^4_{cruise}), into

$\forall\ t : time$
- $(L^{1^*}_{off})\ CurrentSpeed^*(off,t) = 0^*(off)$
- $(L^{4^*}_{cruise}) \geq^*(cruise, Pedal^*(cruise,t), 0^*(cruise)) \Rightarrow$
  $Pedal(cruise,t) = Automatic^*(cruise,t).$

For instance, transition (T_1) is expressed by

- $(T_{1*})\ IgnOn(off) \Rightarrow [(\forall w : st^*)\ (off, w) \in next$
  $\Rightarrow inactive = w \wedge (\exists w' : st^*)\ (off, w') \in next \Rightarrow inactive = w']$,

i.e.,

- $IgnOn(off) \Rightarrow (off, inactive) \in next.$

6. Conclusion and future work

This paper presents a methodology for requirements specification of reactive systems, using hybrid logics, in three stages: firstly the modes and system evolutions are identified (on a standard state-machine), then the functionalities of the system, in each mode, are specified as an algebraic theory (modeled on a states-as-algebras structure), and, finally, the specifications are translated and analyzed in FOL. The combination of the first two stages provides a significant gain in expressivity, when compared to the individual use of classical state machines and algebraic specifications. The comorphism

to FOL allows the (otherwise impossible) adoption of existing mature verification tools.

An interesting direction to pursue is the further exploration of such FOL-specifications as a common support to relate aspects of both machines, namely, through the formalization of the a refinement notion between both machines. Evaluating the initially identified (i-vi) requirements management tasks, this approach increases the verification capability of traceability relations between requirements. Actual full coverage throughout the complete development life-cycle is yet an open issue; a possible approach would be further mapping of the presented methodology to behavioural interface specification languages [HLL$^+$10], which can in turn provide formal code verification. Requirements elicitation is an engineering task preceding the scope of the formalization. Specification and verification of requirements are clearly covered by the presented methodology. Validation against implementation can yet be boosted by the automatic generation of test cases from formal requirements, a current topic of significant research, and clearly a promising expansion of this work.

Acknowledgment: Research partially supported by FCT, the *Fundação Portuguesa para a Ciência e a Tecnologia* through *Centro de Investigação e Desenvolvimento em Matemática e Aplicações* of University of Aveiro and the project MONDRIAN (under the contract **PTDC/EIA-CCO/108302/2008**). A. Madeira is also supported by **SFRH/BDE/33650/2009**, a joint PhD grant by *FCT* and *Critical Software S.A., Portugal.*

References

[Ber02] Daniel M. Berry. Formal methods: the very idea - some thoughts about why they work when they work. *Sci. Comput. Program.*, 42(1):11–27, 2002.

[Bla00] Patrick Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of IGPL*, 8(3):339–365, 2000.

[DF02] R. Diaconescu and K. Futatsugi. Logical foundations of CafeOBJ. *Theor. Comput. Sci.*, 285(2):289–318, 2002.

[Dia08] Razvan Diaconescu. *Institution-independent Model Theory.* Birkhäuser Basel, 2008.

[GB92] Joseph A. Goguen and Rod M. Burstall. Institutions: abstract model theory for specification and programming. *J. ACM*, 39:95–146, January 1992.

[GKS10] Daniel Götzmann, Mark Kaminski, and Gert Smolka. Spartacus: A tableau prover for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 262:127–139, 2010.

[HA09] Guillaume Hoffmann and Carlos Areces. Htab: a terminating tableaux system for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 231:3–19, 2009.

[HKL97] Constance L. Heitmeyer, James Kirby, and Bruce G. Labaw. The SCR Method for Formally Specifying, Verifying, and Validating Requirements: Tool Support. In *ICSE*, pages 610–611, 1997.

[HLL$^+$10] John Hatcliff, Gary T. Leavens, K. Rustan M. Leino, Peter Muller, and Matthew Parkinson. Behavioral interface specification languages. Technical Report CS-TR-09-01a, University of Central Florida, 2010.

[HS08] Christian Hoareau and Ichiro Satoh. Hybrid logics and model checking: A recipe for query processing in location-aware environments. In *AINA*, pages 130–137. IEEE Computer Society, 2008.

[Lan09] Martin Lange. Model checking for hybrid logic. *J. of Logic, Lang. and Inf.*, 18(4):465–491, 2009.

[MHST03] 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.

[MMDB11] Manuel A. Martins, Alexandre Madeira, Răzvan Diaconescu, and Luís S. Barbosa. Hybridization of institutions. Technical report, CCTC, Minho University (submitted to a conference), 2011.

[MML07] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, hets. TACAS'07, pages 519–522, Berlin, Heidelberg, 2007. Springer-Verlag.

[Mos03] Till Mossakowski. Foundations of heterogeneous specification. In *WADT 2002, LNCS*, pages 359–375. Springer, 2003.

[Tar03] A. Tarlecki. Abstract specification theory: An overview. In *Models, Algebras, and Logics of Engineering Software,M. Broy, M. Pizka eds.*, NATO Science Series, Computer and Systems Sciences, VOL 191, pages 43–79. IOS Press, 2003.

[vE02] Jan van Eijck. Hylotab-tableau-based theorem proving for hybrid logics. Technical report, 2002.