Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Sureté de Logiciel et Calcul à Haute Performance

Présentée et soutenue par :

M. GUILLAUME BABIN

le jeudi 6 juillet 2017

Titre :

A formal approach for correct-by-construction system substitution

---

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)

Directeur(s) de Thèse :

M. YAMINE AIT AMEUR

M. MARC PANTEL

Rapporteurs :

M. ALEXANDER ROMANOVSKY, UNIVERSITY OF NEWCASTLE GB

Mme CATHERINE DUBOIS, ENSIIE

M. MICHAEL LEUSCHEL, HEINRICH-HEINE-UNIVERSITAT DUSSELDORF

Membre(s) du jury :

M. DOMINIQUE MERY, UNIVERSITÉ LORRAINE, Président

M. MARC PANTEL, INP TOULOUSE, Membre

Mme ELENA TROUBITSYNA, ABO AKADEMI UNIVERSITY, Membre

M. YAMINE AIT AMEUR, INP TOULOUSE, Membre

Abstract

Safety-critical systems depend on the fact that their software components provide services that behave correctly (*i.e.* satisfy their requirements). Additionally, in many cases, these systems have to be adapted or reconfigured in case of failures or when changes in requirements or in quality of service occur. When these changes appear at the software level, they can be handled by the notion of substitution. Indeed, the software component of the source system can be substituted by another software component to build a new target system. In the case of safety-critical systems, it is mandatory that this operation enforces that the new target system behaves correctly by preserving the safety properties of the source system during and after the substitution operation.

In this thesis, the studied systems are modeled as state-transition systems. In order to model system substitution, the Event-B method has been selected as it is well suited to model such state-transition systems and it provides the benefits of refinement, proof and the availability of a strong tooling with the Rodin Platform.

This thesis provides a generic model for system substitution that entails different situations like cold start and warm start as well as the possibility of system degradation, upgrade or equivalence substitutions. This proposal is first used to formalize substitution in the case of discrete systems applied to web services compensation and allowed modeling correct compensation. Then, it is also used for systems characterized by continuous behaviors like hybrid systems. To model continuous behaviors with Event-B, the Theory plug-in for Rodin is investigated and proved successful for modeling hybrid systems. Afterwards, a correct substitution mechanism for systems with continuous behaviors is proposed. A safety envelope for the output of the system is taken as the safety requirement. Finally, the proposed approach is generalized, enabling the derivation of the previously defined models for web services compensation through refinement, and the reuse of proofs across system models.

Acknowledgments

I would like to express my deep gratitude to Yamine Aït-Ameur and Marc Pantel, my supervisors, for this opportunity as well as for their support, their advice, their trust, their time and their patience. I would also like to thank them for organizing the additional funding that enabled me to complete this thesis and for having enabled me to travel to present my work at conferences and to participate in academic events.

I would like to thank sincerely Catherine Dubois, Michael Leuschel and Alexander Romanovsky who have reviewed this thesis. Thank you for spending the time to read this document in details and provide valuable feedback and constructive suggestions. I would also like to thank them as well as Dominique Méry, Elena Troubitsyna and Laurent Voisin for accepting to be part of the defense committee.

I thank Arnaud Dieumegard with whom I enjoyed working on my first research project and who has been a great example as Ph.D. student.

I am particularly grateful for the assistance given by Aurélie Hurault in helping me to candidate to wonderful internships.

I thank Shin Nakajima who supervised my stay at the National Institute of Informatics in Tokyo for the opportunity and for his perspective on my work.

I wish to thank Neeraj Kumar Singh for his collaboration and his advice.

I would like to thank Marc Pantel, Xavier Crégut, Joseph Gergaud and Daniel Ruiz for offering me the opportunity to teach and for the many interesting ensuing discussions.

I wish to acknowledge the help provided by Sylvie Eichen, Sylvie Armengaud-Metche, Annabelle Sansus and Muriel De Guibert who were always pleasant, efficient and helping in all administrative matters.

I thank all the people working at ENSEEIHT and IRIT, the members of the ACADIE team, and especially the Ph.D. students Arnaud, Florent, Ning, Faiez, Soukayna, Mathieu, Florent, Kahina, and Alexandra.

Finally, I thank my parents who have always supported me, encouraged me and believed in me.

Contents

List of Figures

List of Tables

List of Models

Introduction

Context

Nowadays, rigorous development methods grounded in mathematical and logical foundations are mature enough to support the development of complex systems, using either pure software, pure hardware or mixing software and hardware parts. Moreover, it is well accepted that these rigorous methods allow increasing the quality of the developed complex systems but also of the development processes that lead to the design of these systems.

Formal methods have proved useful in many safety critical application domains and industries like aeronautics, space, automotive and rail transportation, medical systems or energy production. Mature tool suites supporting such formal methods are now available. They assist in the system design through complexity management (using refinement/abstraction, composition/decomposition). They provide support tools and techniques to understand systems (with simulation and animation), identify design errors (with model-checking and tests) and/or demonstrate correctness (with proofs). Several tooled framework enabling formal methods and techniques have been developed to handle system development or part of it. Specification, validation, verification, simulation, design, *etc.* are some of the activities targeted by formal methods and associated framework. One key enabler for the large scale use of formal methods is the identification of domain, problem or application families and associated verification strategies that ease the application of formal methods in realistic industrial applications.

One of the important problem family studied in system engineering relates to system evolution or system changes during its lifetime (for example to integrate updates or manage and react to failures). Handling the changes of a system is a key requirement particularly in the case of adaptive, self-healing, autonomous, or reconfigurable systems and in other situations like maintenance or redundancy. These changes may occur in different cases like changes in the specification, the environment, quality of service, running platform, *etc.* At this level, fundamental questions related to recording system changes arise:

- *What are the preserved system properties?*
- *What are the lost system properties?*
- *What are the new properties of the system after changes?*

Handling system evolution requires to answer the above mentioned questions. When systems are critical systems with hard safety and dependability requirements and with certification, it is needed to set up verification and validation techniques that allow developers and customers to have the appropriate confidence on the developed system. Formal methods have proved useful to fulfill such requirements.

Therefore, when systems are formally modeled, it becomes possible to set up a formal reasoning allowing developers to manage system evolution using formal modeling techniques.

In this thesis, we focus on the study of the critical system evolution problem family, when formally modeled, that may occur either at design time (during system development) or at runtime (when the system runs). We claim that various system changes can be formally modeled by a system substitution operation which consists in substituting a system by another one preserving the original system state. The provided results will enable a more efficient development based on formal methods of this kind of systems and provide a better scalability for the use of formal methods.

Objectives of the thesis

As mentioned above, in this thesis we address the problem of handling system changes and updates at design time and runtime. A system substitution operation is proposed to handle various types of system changes. We have chosen to model the considered systems as state-transition systems and to use the Event-B refinement and proof based formal method as a supporting method for all the developments we have achieved.

The goal of our work is to define system substitution by a generic development operation that records system changes from a source system to a target system. This generic operation thus allows to ease the development of this problem family. To reach this goal, we have identified the following objectives:

- Define a formal framework to model both system specification and implementations of such evolutive systems.

- Identify the system substitution operation between systems implementing (refining) a common specification and the corresponding properties (proof obligations) of that operation. Provide a formalization for this operation.

- Handle the case of substitution at runtime or at design time (cold or hot substitution).

- Address degraded, upgraded or equivalent modes of the target system after substitution.

- Study the case of substitution of a system by itself (self-$\star$ systems, autonomous systems), or by an update of the source system with new parts issued from another system, or by a new system.

- Consider different types of systems candidate for substitution: discrete event-based systems and hybrid systems with continuous behavior.
- Offer the appropriate set of proof techniques to handle both discrete and continuous proofs associated with the studied systems.

Contributions

As mentioned above, the main objective of our work is to define a formal model for the system substitution problem family in different situations. We use the Event-B refinement and proof-based method to model both the systems and the proposed system substitution operation. Event-B enables us to benefit from refinement and correctness proofs, all supported by the Rodin Platform.

In our approach, systems are modeled as state-transition systems. We are concerned with safety properties modeled as invariants. These properties need to be preserved during and after system substitution. Our contributions consists in the following:

- Definition of a generic framework for system substitution together with the identification of the properties to ensure the preservation of the safety requirements of the source system.
- Use of the proposed substitution mechanism for systems characterized by discrete event systems. In this case, we consider instantaneous system substitution. The particular case of web services compensation has been studied.
- Use of the proposed substitution mechanism for hybrid systems characterized by continuous behaviors. In this case, we consider non-instantaneous system substitution. The case of a continuous function characterizing system behaviors is considered.
- Formalization of system substitution as a generic operator that manipulates systems, states and transitions. The relevant properties of this operator are also formalized. This operator is used for a class of systems that instantiate the proposed generic systems descriptions.

These contributions will be detailed in the next chapters of this thesis.

Thesis outline

This thesis is organized as follows.

The first part is devoted to the state of the art. Refinement and proof-based formal methods with explicit state definition are introduced in Chapter 1. A focus on the chosen Event-B method is provided.

The second part presents our contributions for system substitution. It shows how the proposed approach applies for substitution of systems described either by discrete or continuous behaviors and how it generalizes to a class of systems.

- The generic framework for system substitution we have defined is presented in Chapter 4. The key concept of *horizontal invariant* is introduced. It models the relation between system states before and after system substitution. Then, the proposed system substitution approach is deployed in two situations. (Related publications [1], [6])

  1. First, application to discrete systems is addressed in Chapter 5. The case of web services compensation is used to illustrate how our approach for system substitution handles web services compensation at runtime. (Related publications [2], [8])
  2. Second, we studied hybrid systems whose behavior is characterized by the integration of both discrete and continuous behaviors modeled with continuous functions. Again, in Chapter 7 the proposed system substitution operator is set up on such systems. Specific features related to correct modeling of such systems with Event-B are given before in Chapter 6. (Related publications [3], [5], [7], [9])

- Finally, a generalization of our approach is presented in Chapter 8. The approach considers systems (state-transition systems) as objects manipulated by the proposed generalized system substitution operation. (Related publications [4], [10]).

Last this thesis ends by a conclusion and a review of the perspectives we have identified.

Publications related to the thesis

The following contributions were accepted and published in conferences and journals.

[1] G. Babin. "A formal approach for correct-by-construction system substitution". In: *The Tenth European Dependable Computer Conference (EDCC) 2014 – Student Forum*. 2014.

[2] G. Babin, Y. Aït-Ameur and M. Pantel. "Formal Verification of Runtime Compensation of Web Service Compositions: A Refinement and Proof Based Proposal with Event-B". In: *IEEE International Conference on Services Computing (SCC)*. 2015.

[3] G. Babin, Y. Aït-Ameur, S. Nakajima and M. Pantel. "Refinement and Proof Based Development of Systems Characterized by Continuous Functions". In: *Dependable Software Engineering: Theories, Tools, and Applications (SETTA)*. 2015.

[4] G. Babin, Y. Aït-Ameur and M. Pantel. "Correct Instantiation of a System Reconfiguration Pattern: A Proof and Refinement-Based Approach". In: *IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*. 2016.

[5] G. Babin, Y. Aït-Ameur, N. K. Singh and M. Pantel. “Handling Continuous Functions in Hybrid Systems Reconfigurations: A Formal Event-B Development”. In: *5th International Conference on Abstract State Machines, Alloy, B, TLA, VDM (ABZ)*. 2016.

[6] G. Babin, Y. Aït-Ameur and M. Pantel. “A generic model for system substitution”. In: *Trustworthy Cyber-Physical Systems Engineering*. 2016. Ed. by Alexander Romanovsky and Fuyuki Ishikawa.

[7] G. Babin, Y. Aït-Ameur, N. K. Singh and M. Pantel. “A System Substitution Mechanism for Hybrid Systems in Event-B”. In: *18th International Conference on Formal Engineering Methods (ICFEM)*. 2016.

[8] G. Babin, Y. Aït-Ameur and M. Pantel. “Web Service Compensation at Runtime: Formal Modeling and Verification Using the Event-B Refinement and Proof Based Formal Method”. In: *IEEE Transactions on Services Computing – Special Issue on Advances in Web Services Research*. 2017.

The following contributions were selected by the conference scientific board and submitted to special issues of journals. They passed the first review steps and are now under revision for the second step.

[9] G. Babin, Y. Aït-Ameur, N. K. Singh and M. Pantel. “Handling Continuous Functions in Hybrid Systems Reconfigurations: A Formal Event-B Development”. In: *Science of Computer Programming – Selected papers – ABZ 2016 – Under revision after first review*. 2017.

[10] G. Babin, Y. Aït-Ameur and M. Pantel. “Correct Instantiation of a System Reconfiguration Pattern: A Proof and Refinement-Based Approach”. In: *Journal of Software: Evolution and Process – HASE 2016 – Under revision after first review*. 2017.

Complete details of these references are available on page 135.

Part I

Background

1 System modeling with Event-B: a correct-by-construction method

This thesis targets the modeling and verification of systems composed of parts that can change during time, either offline or online. These changes of systems part can be modeled nicely using system state changes. We thus decided to rely on state-transition systems as model of computations, on the Event-B method and the Rodin Platform as support for the system modeling and requirement satisfaction proofs structured using refinements. We will first summarize these formal techniques.

1.1 Models of systems

Transition systems have been identified as an appropriate generic model for systems. They support the definition of systems and their behaviors and they allow developers to reason on their execution traces. One of the design methodologies associated with transition systems consists in describing a sequence st_i of such systems where st_i refines st_{i-1}. The refinement introduces more and more details growing from an abstract system to a concrete one. Moreover, we target the definition of correct systems that are possibly parameterized. Therefore, it is required to prove the correctness of the designed models beyond (partial) testing or bounded model checking.

Several formal methods to define and model such systems have been proposed in the literature. The first class of formal methods is based on the definition of process algebras. Examples of such modeling languages are CCS [Mil80] or LOTOS [EVD88; ISO89]. These techniques do not offer well-accepted refinement operations. So we did not consider them in our work.

The second class of formal methods is the so-called state-based formal methods. These methods have drawn the attention of several researchers. They are based on the definition of systems states (through a set of state variables) and transitions (from a state to another) equipped in general with pre-conditions and post-conditions [Hoa69] to offer reasoning capabilities. Moreover, this formal model has been associated to a refinement relation allowing the definition of a sequence of models linked by this relation. Among these methods we can cite Z [Spi92; ISO02], VDM [BJ78], B [Abr96], TLA$^+$ [Lam02], Event-B [Abr10] and Statecharts [Har87]. In the recent developments, these methods have been associated to several model checking techniques and tools offering capabilities for model verification and/or animation. Examples of such model checkers are NuSMV [Bur+92], CADP [Gar+13], PROMELA/SPIN [Hol04], ProB [LB03] and TINA [BV06].

A third class of formal methods relates to the so-called "higher-order formal methods". Thanks to their higher order characteristics, these methods offer the capability to describe system models and the associated verification procedure in a uniform setting. They could be used at a "meta" level: they would need an encoding of the notions of state and transition using higher-order functions. Such methods are Isabelle/HOL [NPW02], PVS [ORS92] or Coq [BC04; The16].

In order to benefit from a methodology based on the native notions of state, transition, refinement, proofs and the availability of a powerful supporting tool (the Rodin Platform), we have chosen the Event-B formal method to express our models and prove the associated properties.

The Event-B method [Abr10] is a recent evolution of the B method [Abr96]. This method is based on the notions of pre-conditions and post-conditions from Hoare [Hoa69], the weakest pre-condition from Dijkstra [Dij97] and the substitution calculus [Abr96]. It is a formal method based on mathematical foundations: first-order logic and set theory.

1.2 Event-B models

An Event-B model is characterized by a set of variables, defined in the `Variables` clause that evolve thanks to events defined in the `Events` clause. It encodes a state-transition system where the variables represent the state and the events represent the transitions from one state to another. During the execution, events are interleaved (*i.e.* at any time, only one event is executed).

An Event-B model is made of several components of two kinds: machines and contexts. The machines contain the dynamic parts (states and transitions) of a model whereas the contexts contain the static parts (axiomatization and theories) of a model. A machine can be refined by another one, and a context can be extended by another context. Moreover, a machine can see one or several contexts.

A context is defined by a set of clauses (Model 1.1) as follows.

- `Context` represents the name of the component that should be unique in a model.
- `Extends` declares the context(s) extended by the described context.
- `Sets` describes a set of abstract and enumerated types.
- `Constants` represents the constants used by a model.
- `Axioms` describes, in first-order logic expressions, the properties (definitions) of the attributes declared in the `Constants` and `Sets` clauses. Types and constraints are described in this clause as well.
- `Theorems` are logical expressions that can be deduced from the axioms.

```
Context ctxt_id_2
Extends ctxt_id_1
Sets s
Constants c
Axioms A(s,c)
Theorems T_c(s,c)
End
```

```
Machine machine_id_2
Refines machine_id_1
Sees ctxt_id_2
Variables v
Invariants I(s,c,v)
Theorems T_m(s,c,v)
Variant V(s,c,v)
Events
  Event Initialisation ≙
    Begin
      v :| D(s,c,x,v')
    End
  Event evtr ≙
    Refines evt
    Any x
    Where G(s,c,v,x)
    Then v :| BA(s,c,v,x,v')
    End
End
```

Model 1.1 – Structures of Event-B contexts and machines

Similarly to contexts, machines are defined by a set of clauses (Model 1.1).

- `Machine` represents the name of the component that should be unique in a model.
- `Refines` declares the machine refined by the described machine.
- `Sees` declares the list of contexts imported by the described machine.

- `Variables` represents the state variables of the model of the specification. Refinements may introduce new variables in order to enrich the described system.

- `Invariants` describes, using first-order logic expressions, the properties of the variables declared in the `Variables` clause. Typing information, functional and safety properties are usually given in this clause. These properties shall remain true at all times. This means that the invariants must hold after the initialization and that events (more precisely their actions) must preserve them. This is enough to guarantee that the invariants always hold by means of mathematical induction.

  It also expresses the gluing invariant required by each refinement.

- `Theorems` defines a set of logical expressions that can be deduced from the invariants and the context(s). They do not need to be proved for each event, contrary to the invariants.

- `Variant` introduces a natural number or finite set that will be used to guarantee termination properties.

- `Events` defines all the events (transitions) that can occur in a given model. Each event is characterized by its guard and is described by a body of actions. Each machine must contain an *Initialisation* event. The events occurring in an Event-B model affect the state described in the `Variables` clause.

  An event consists of the following clauses (Model 1.1):

  - `Refines` declares the list of events refined by the described event.
  - `Any` lists the parameters of the event.
  - `Where` expresses the guard of the event. An event can be fired (triggered) when its guard evaluates to true. If several guards evaluate to true, only one can be fired with a non-deterministic choice.
  - `Then` contains the actions of the event that are used to modify variables.

  In order to model termination properties, events are marked as:

  - `ordinary`: there is no restriction regarding the variant,
  - `convergent`: the variant must decrease,
  - `anticipated`: the variant must not increase. This is intended to be used with refinement.

Event-B offers three kinds of actions (substitutions):

- *assignment* (`x` := `E`) where the variable becomes equal to the value of a particular expression. This action is deterministic.

  Example: `x` := `4`

- *choice* (`x :∈ S`) where the variable takes a value from a set, in a non-deterministic manner.

  Example: `x :∈ ℕ\{2}`
  where the variable x takes as value any natural number other than 2.

- *before-after* predicate (`x :|` BA(`x,x'`)), is the more general form of action. The new values of the variables become such that the given before-after predicate holds. The future values are quoted, the current ones are not. This is the more powerful notation since it can express all the others. It is compulsory when expressing relations between the future values of multiple variables in an action, as otherwise actions are independent. However, by adding parameters with guards, the first form := is sufficient.

  Example: `x,y :| x'>x ∧ x'+y' = 5`
  It asserts that x and y take any values such that x becomes greater than its previous value and that the sum of the new values of x and y is equal to 5.

1.3 Proof obligation rules

Proof obligations (PO) are associated with any Event-B model to express the correctness of the developments and refinements. They must be proved to ensure the correctness of the model.

The rules for generating proof obligations follow the substitutions calculus [Abr10; Abr96], close to the weakest precondition calculus of Dijkstra [Dij97]. In order to define proof obligation rules, we use the notations defined in Model 1.1 where s denotes the seen sets, c the seen constants, and v the variables of the machine. Seen axioms are denoted by $A(s, c)$ and theorems by $T_c(s, c)$, whereas invariants are denoted by $I(s, c, v)$ and local (event-specific) theorems by $T_m(s, c, v)$. For an event, the guard is denoted by $G(s, c, v, x)$ and the action is denoted by the before-after predicate $BA(s, c, v, x, v')$. The prime notation v' denotes the variable v after action execution.

Table 1.1 – Examples of proof obligations for an Event-B model

Theorems	$A(s, c) \Rightarrow T_c(s, c)$ $A(s, c) \wedge I(s, c, v) \Rightarrow T_m(s, c, v)$	(a) (b)
Invariant preservation	$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \wedge BA(s, c, v, x, v') \Rightarrow I(s, c, v')$	(c)
Event feasibility	$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \Rightarrow \exists v'.BA(s, c, v, x, v')$	(d)
Natural variant	$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \Rightarrow V(s, c, v) \in \mathbb{N}$	(e)
Variant progress	$A(s, c) \wedge I(s, c, v) \wedge G(s, c, v, x) \wedge BA(s, c, v, x, v')$ $\Rightarrow V(s, c, v') < V(s, c, v)$	(f)

Table 1.1 shows the main obligation rules associated to an Event-B model.

- The *theorem proof obligation rules* (a) and (b) ensure that a proposed context theorem (a) or machine theorem (b) is indeed correct: it can be deduced from the axioms and the invariants.
- The *invariant preservation proof obligation rule* (c) ensures that each invariant in a machine is preserved by each event.
- The *feasibility proof obligation rule* (d) ensures that a non-deterministic action is feasible.
- The *natural variant proof obligation rule* (e) guarantees that under the guards of each convergent or anticipated event, a proposed numeric variant is indeed a natural number.
- The *variant proof obligation rule* (f) states that each convergent event decreases the proposed numeric variant.

There are other rules for generating proof obligations to prove the correctness of refinement. The complete definitions are given in [Abr10].

1.4 Semantics

The new aspect of the Event-B method [Abr10], in comparison with classical B [Abr96], is related to the semantics. Indeed, the events of a model are atomic events of a state-transition system. The semantics of an Event-B model is a trace-based semantics with interleaved events. A system is characterized by the set of licit traces corresponding to the fired events of the model which respect to the described properties. The traces define a sequence of states that may be observed by properties. All the properties will be expressed on these traces.

1.5 Refinement

The refinement operation [AH07] offered by Event-B enables stepwise model development. A state-transition system is refined into another state-transition system with more and more design decisions while moving from an abstract level to a less abstract one. A refined machine is defined by adding new events, new state variables and a gluing invariant. Each event of the abstract model is refined in the concrete model by adding new information expressing how the new set of variables and the new events evolve. All the new events appearing in the refinement refine the *skip* event (which is the event that does nothing and can occur any time). Refinement preserves the proved properties and therefore it is not necessary to prove them again in the refined transition system, usually more detailed. This help keeping the proof sizes reasonable by distributing the proof effort along the refinement tree.

In order to prove the correctness of the development, it is necessary to prove the correctness of the various refinements it contains. The following proof obligations are the two key proof obligations.

- Guard strengthening: a concrete event must be enabled only if the abstract event is enabled.
  For each abstract i-th guard G_i^A,

  $$A \wedge I^A \wedge I^C \wedge G^C \wedge W \Rightarrow G_i^A$$

  where, as a reminder, A denotes the conjunction of the axioms, I the invariants, G the guards, W the witnesses (predicates linking concrete and abstract variables) and BA before-after predicates (actions); and $\cdot^A$ relates to the abstract machine while $\cdot^C$ relates to the concrete one.

- Action simulation: if an abstract event's action assigns a value to a variable that is also declared in the concrete machine, it must be proven that the abstract event's behavior corresponds to the concrete behavior.

  $$A \wedge I^A \wedge I^C \wedge G^C \wedge W \wedge BA^C \Rightarrow BA_i^A$$

Remark Note that many different refinements may refine the same given abstract machine. Each refinement machine corresponds to a possible behavior, implementation or concretization of the abstract machine. Thus, several candidate refinements are offered for a given abstract machine. This will be used in later chapters to characterize the set of correct systems that behave as described by an abstract system description.

The Event-B method proved its capability to represent event-based systems like railway systems, embedded systems or web services. Moreover, complex systems can be gradually built in an incremental manner by preserving the initial properties thanks to the preservation of a gluing invariant.

1.6 Liveness & deadlock

1.6.1 Liveness properties

The built-in facilities of Event-B are mainly oriented towards guaranteeing safety properties (absence of bad states) thanks to invariants preservation. However, it is also possible to verify some liveness properties:

- within Event-B where LTL formulas can be directly encoded [HA11] although it is not really practical for large formulas.

- using external tools such as the model checking ProB which can verify LTL formulas on bounded Event-B models [PL10].

It is important to note that, contrary to safety properties, liveness properties are not systematically preserved by refinement.

1.6.2 Deadlock-freeness

We define a *deadlock* as a state in which none of the events are possible: the system will not progress anymore because none of the transitions are enabled.

We can express the *deadlock-freeness* invariant (DLF) as the disjunction of the guards of all events other than the initialization:

$$DLF = \bigvee_{\text{event } e} \left(\bigwedge_{\text{guard } G_i \text{ of } e} G_i \right)$$

By proving that DLF is a theorem, we can demonstrate that the machine will never deadlock. Indeed, we prove that, at any time, at least one event has all its guards evaluating to *true*. Therefore, at least one event is possible (enabled transition) at any time.

It is also possible to consider the deadlock-freeness of a subset of events.

1.7 Tools

The main tool available for conducting Event-B based developments is the Rodin Platform[1] [Abr+10]. This is an integrated development environment equipped with contexts and machines editors, a proof obligation generator, automated provers and interactive proving capabilities.

Additionally, a wide range of plug-ins are available, which can for instance extend the modeling (for instance with *theories*) or proving capabilities (such as the model checker ProB or the use of SMT solvers).

Animation It is also possible to instantiate the models within the Rodin Platform and to animate them. This is very useful to check with domain engineers if the specification produces the intended behaviors, and to verify if the models, additionally to not violate invariants, can actually exist.

1.8 Uses of reals

In order to model cyber-physical systems where the continuous world meets the discrete world, time is a mandatory feature that must be modeled as a continuous variable. Mathematical real numbers are thus needed to model time.

1.8.1 The Theory plug-in

A recent evolution of the Event-B method makes it possible to extend it with theories similar to algebraic specifications. In the Rodin Platform, this evolution is provided by the *Theory* plug-in [Abr+09; BM13; Hoa+17].

[1] http://www.event-b.org/

Several theories have been written and are available as a Standard Library[2] which contains 3 groups of theories:

- *Basic* which includes theories *BinaryTree* (binary trees), *BoolOps* (boolean operators), *List* (inductive lists), *PEANO* (inductive natural numbers), *SUMandPRODUCT* (generalized sum and product) and *Seq* (sequences)

- *RelationOrder* which includes theories *Connectivity* (graph connectivity), *FixPoint* (lower & upper fixpoints), *Relation* (ordering relations: transitivity, reflexivity, . . .), *Well_Fondation* (well-founded relations), *closure* (relational closure), *complement* (complement & conjugate) and *galois* (galois connections)

- *Real* which includes a theory *Real* of mathematical real numbers

According to the documentation[3], a theory definition can include the following elements.

- Datatypes which are defined by providing the types on which they are polymorphic, a set of constructors one of which has to be a base constructor. Each constructor may or may not have destructors.

- Operators that can be defined as predicate or expression operators. An expression operator is an operator that "returns" an expression, an example existing operator is *card*. A predicate operator is one that "returns" a predicate, an example existing predicate operator is *finite*.

- Axiomatic definitions that are defined by supplying the types, a set of operators, and a set of axioms.

- Rewrite rules which are one-directional equalities that can be applied from left to right.

- Inference rules that can be used to infer new hypotheses, split a goal into sub-goals or discharge sequents.

- Polymorphic theorems that can be defined and validated once, and can then be imported into sequents of proof obligations inside a proof if a suitable type instantiation is available.

In order to validate the extension, proof obligations are generated to ensure soundness of extensions. This includes, proof obligations for validity of inference and rewrite rules, as well as proof obligations to validate operator properties such as associativity and commutativity.

[2]`http://wiki.event-b.org/index.php/Theory_Plug-in#Standard_Library`
[3]`http://wiki.event-b.org/index.php/Theory_Plug-in#Capabilities`

1.8.2 Theory *Real*

We use the theory *Real* (Appendix A, page 140), written by Abrial and Butler, which models mathematical real numbers. This theory provides:

- 1 datatype `REAL`
- 13 operators: `plus` ($+$), `minus` (unary $-$), `mult` ($\times$), `sub` ($-$), `inv` ($\frac{1}{\cdot}$), `leq` ($\leq$), `smr` ($<$), `gtr` ($>$), `cnt` (point-wise function continuity), `inf` (infimum), `sup` (supremum) as well as `zero` and `one`
- 24 axioms that define the semantics of the operators
- 18 interactive rewrite rules for use in proofs

The theory *Real* is minimal which makes it mathematically elegant, however it makes the proofs very long because everything has to be decomposed on very simple propositions in order to apply the axioms. That is why, during the development of the models, we defined a context *C0_reals* (Appendix C, page 221) with 43 additional theorems selected from repetitive interactive proofs. It was crucial in managing the time spent on proving models. It contains fairly basic theorems such as:

- $a + c \leq b + c \Leftrightarrow a \leq b$
- $a \times (-1) = -a$
- $\forall x \in [a, b] \quad f(x) = g(x) \Rightarrow$ (f continuous on [a,b] $\Leftrightarrow$ g continuous on [a,b])

1.8.3 Casting

However, because neither implicit type conversion nor operator overloading are available in Event-B, we have defined a *cast* function that maps naturals to their representation as positive reals, in order to be able to write expressions such as $n \times \delta t$ where $n \in \mathbb{N}$ and $\delta t \in \mathbb{R}$.

The function *cast* has been defined inductively on naturals. Several theorems such as the fact that *cast* is an order isomorphism from (`NAT`, `<=`) to ($\mathtt{REAL}_{|\mathbb{N}}$, `leq`) needed to be proved.

Note that the context *C3_cast* (Model 1.2 & Appendix C, page 233) extends the context *Nat* (page 232), written by Thái Sơn Hoàng, which contains the induction theorem.

1.8.4 Reals and floats

Our developments rely on mathematical real numbers. We decided to stop the development before the translation to machine numbers (floating-point or fixed-point numbers) that must be introduced in further refinements if we target the translation to realistic embedded software. This topic is thus out of the scope of our work and we do not need a model of floating-point or fixed-point computation. This could also have been conducted using the Theory plug-in.

```
Context C3_cast Extends C0_reals, Nat
Constants cast
Axioms
  axm01: cast ∈ ℕ→ℝ+        // type
  axm02: cast(0) = zero     // initial case
  axm03: ∀a · a ∈ ℕ ⇒(cast(a+1) = cast(a) plus one)   // induction case
Theorems
  …
  thm11: ∀a,b · (a ∈ ℕ ∧ b ∈ ℕ) // equiv. over '<'
            ⇒(a < b ⇔smr(cast(a),cast(b)))
  thm12: ∀a,b · (a ∈ ℕ ∧ b ∈ ℕ) // equiv. over '='
            ⇒(a = b ⇔cast(a) = cast(b))
  thm13: cast ∈ ℕ ⤖ cast[ℕ] // cast is a bijection
  …
End
```

Model 1.2 – Definition and properties of the *cast* function

2 System substitution

During a system development and execution, some operations (*e.g.* maintenance) or development actions (*e.g.* upgrade) involve mechanisms that correspond to changes in system parts that can be represented by sub-system substitution.

2.1 System substitution: definition and characteristics

System substitution is an operation defined as the capability to replace a source system by another one (target system) that preserves the specification of the source one. This operation may occur in different situations like failure management, maintenance, reconfiguration, adaptive systems or autonomous systems. When substituting a system at runtime, a key requirement is to identify the correct state of the target system that restores the identified state of the source system. The correctness of the state restoration relies on the definition of safety properties for system substitution. Our main concern consists in identifying the relevant properties required to be proven in order to assert the correctness of the system substitution.

2.1.1 Persistence of the system state after substitution: Cold and Warm start

One first characteristic is the persistence of the state after substitution, usually named cold or warm start. It characterizes the restored state in the substitute system.

Cold start, tagged as *Static substitution*, means that the substitute system will start from its initial state without any data nor state variables values originated from the state where the original system was halted.

Warm start, tagged as *Dynamic substitution*, means that the substitute system will recover as much data and state variable values as possible coming from the state where the original system was halted. In other words, when a system is halted in order for a second system to replace it, the second system is positioned in a state that is functionally identical (or as close as possible) to the state of the first system when it was stopped. This enables the second system to continue the task the first system was doing (almost) without interruption, as seen from outside of the system.

2.1.2 Identical, included or disjoint sets of state variables

If we assume that we have two systems – a source and a target – that we model as state-transition systems where their states are represented as a set of state variables, then we can distinguish three cases during the substitution of the source system by the target system.

- The sets of states variables are identical. This situation means that the original (source) and the substitute (target) systems represent the same system. The effect of the substitution is to restore a new state, correct with respect to the represented system substitution properties, after substitution. This situation usually occurs in case of maintenance or autonomous systems, self-healing systems.
  Example: an e-commerce website that would be replaced by a website offering the same services.

- The sets of states variables are partially shared. In this case, part of the original system state variables are restored in the substitute system, and the substitute system introduces new state variables that describe new behaviors.
  Example: an e-commerce website that would be replaced by a smartphone application and a new website.

- The sets of states variables are disjoint. Disjointness implies that the original and substitute systems are independent *i.e.* the substitute system is a new system. The repair or substitution transfers the control to a completely new substitute system.
  Example: an e-commerce website that would be replaced by a smartphone application.

2.1.3 Equivalent, upgraded or degraded substitution

Another characteristic relates to the behavior of the substitute system and the associated quality of the substitution. Several substitute systems may offer different functionalities and have different behaviors. Three cases have been identified. The substitute system may be equivalent to the original system, may upgrade it (enhance it) or may degrade it.

- *Equivalence* means that the original system properties are preserved *i.e.* the substitute system offers the same functionalities, but may differ from quality of service point of view.
  Example: an e-commerce website that would be replaced by a website selling the same set of products.

- *Upgrade* is stronger than equivalence. The substitute system provides the same functionalities as the original system, but it also provides more functionalities.
  Example: an e-commerce website that would be replaced by a website selling more products than in the original website.

- *Degradation* is weaker than equivalence. The substitute system provides fewer functionalities than the original system.
  Example: an e-commerce website that would be replaced by a website selling only a subset of products available in the original website.

2.1.4 Instantaneous or delayed (deferred) substitution

The nature of the system can impact how the substitution will behave. In a discrete system, the substitution can be instantaneous. In that case, substitution is seen as an atomic operation: at an instant, a system was running, at the next instant, another system is running.

However, for cyber-physical systems with continuous behaviors modeled over continuous time, it is not possible to shut down such a system instantly. The system needs to be shut down over a period of time, while a substitute system is prepared to take over. The substitution is more complex in this case, as for some period of time, both systems are running, and the substitution cannot be considered as an atomic operation.

2.1.5 Static or dynamic set of substitutes

One can imagine that the set of substitutes may evolve. A substitute system can be added or removed from the set of substitutes. The set of substitutes would then be considered *dynamic* as opposed to a fixed set of substitutes which would be designated as *static*.

2.1.6 Centralized or distributed system substitution

In a centralized architecture, there exists a unique controller that can decide whether or not to trigger a substitution on the components of the system. In a distributed architecture, each system will individually decide if and when it is appropriate to trigger a substitution based on available information (possibly obtained after communicating with neighbor systems).

2.1.7 Local or global invariant

In the case of a single system, the system tries to maintain an invariant involving its local state. We can also envision more complex architectures where a set of systems try to preserve a global invariant involving a collection of their states variables.

2.2 Studied systems

The systems addressed by our approach are formalized by state-transition systems [Arn88], which proved to be useful to model various kinds of systems and particularly hybrid systems [Alu11] or cyber-physical systems [LS14]. In particular, controllers are modeled with state-transition systems.

A system is characterized by a state that may change when a transition occurs. A state is defined as a set of pairs $(variable, value)$. The values of a given variable are taken in a set of values satisfying safety properties expressed within invariants (Kripke structure). A transition characterizes a state change, through updating of variable values.

Figure 2.1 presents the abstract model of the systems we consider. After being initialized, these systems run (*progress*) until they fail or they are stopped.

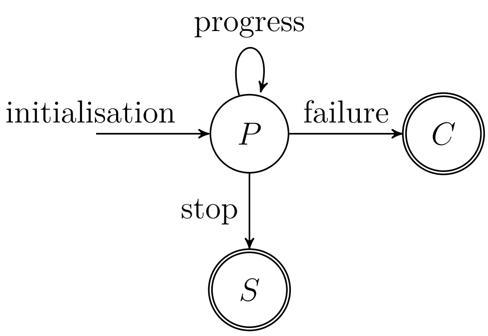


Figure 2.1 – System abstraction

By combining two basic systems into a global system as in Figure 2.2, the second system (here in blue, with elements $\Box_T$) can replace the first system (here in red, with elements $\Box_S$) when it fails.

We can abstract the global system of Figure 2.2 by the system of Figure 2.3.

The first model (Figure 2.1) is also an abstraction of the last model of Figure 2.3.

From this point forward, we will consider systems with behaviors corresponding to the ones of Figure 2.4: a system is initialized, then it evolves (*progress*), relying on state changes. A failure (*fail*) can occur during state change. The system may then be repaired (*repair*), or isolated (*complete failure*).

Below, we show how such transition systems are modeled with the Event-B method.

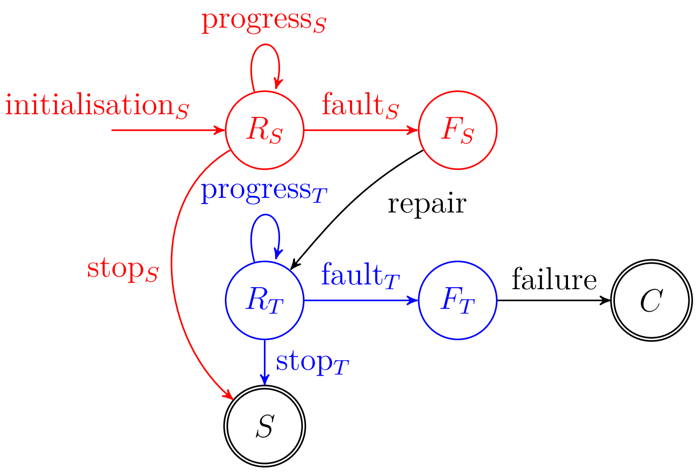


Figure 2.2 – Combination of systems

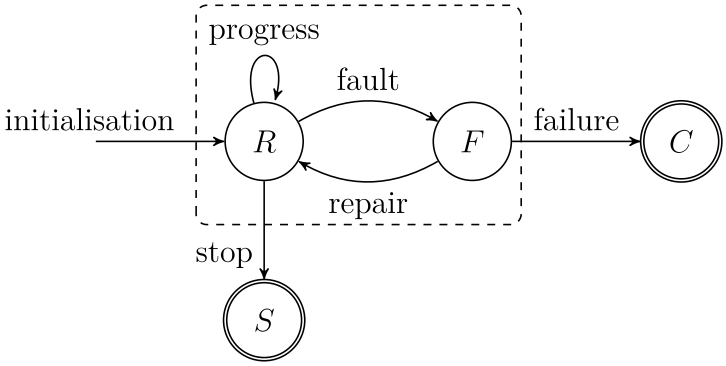


Figure 2.3 – System abstraction, with failure

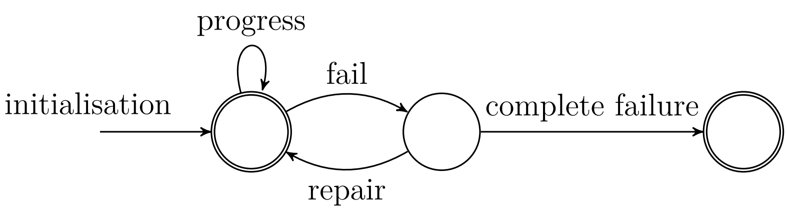


Figure 2.4 – Studied system behavior pattern

2.2.1 Specification of studied systems

When the studied systems are described as state-transition systems, they are modeled using Event-B as follows.

- A set of variables, in the Variables clause is used to define system states. The Invariants clause describes the relevant properties of these variables.
- An Initialisation event determines the initial state of described system by assigning initial values to the variables.
- A set of (guarded) events defining transitions is introduced. They encode transitions and record variable changes.

A state transition system (where the variables clause defines states and the events clauses define transitions, see Model 2.2) is described in an Event-B machine Spec. This machine sees the context C0 (see Model 2.1) from which it borrows relevant definitions and theories.

```
Context C0
Sets s
Constants c
Axioms A(s, c)
End
```

Model 2.1 – Context *C0*

```
Machine Spec
Sees C0
Variables v_A
Invariants I_A(s, c, v_A)
Events
  Event  Initialisation  ≙
    Begin
      v_A :| D_A(s, c, v'_A)
    End
  Event Evt ≙
    Any
      x_A
    Where
      G_A(x_A, s, c, v_A)
    Then
      v_A :| BA_A(x_A, s, c, v_A, v'_A)
    End
End
```

Model 2.2 – Machine *Spec*

```
Machine SysS
Refines Spec
Sees C0
Variables v_S
Invariants I_S(s, c, v_S, v_A)
Variant VN_S
Events
  Event  Initialisation  ≙
    Begin
      v_S :| D_S(s, c, v'_S)
      VN_S :| VN_S_InitValue
    End
  Event s_evt ≙
    Any x_S
    Where
      G_S(x_S, s, c, v_S)
    Then
      v_S :| BA_S(x_S, s, c, v_S, v'_S)
    End
  ...
  Event Evt Refines Evt ≙...
End
```

Model 2.3 – Machine *SysS*

2.2.2 Refinement of studied systems

The previously defined state-transition system may be defined at a given abstraction level. It constitutes a system specification. Several candidate systems S_i may refine (implement) the same specification *Spec*. These implementations are more concrete state-transition systems that refine an abstract one. Model 2.3 shows such a refinement. A new set of variables and events is introduced that refines the abstract model.

Refinement relies on the definition of a gluing invariant. The verification of the correctness of this refinement ensures that the refined system is a correct implementation of the specification it refines.

Definition of substitute systems We have chosen to use the refinement relationship in order to characterize all the substitute systems. If we consider a system characterized by an original specification, then all the systems that refine this specification are considered as potential substitutes. Obviously, we are aware that these refining systems are different and may behave differently, but we are sure that these behaviors include the one of the refined system.

2.3 Formal methods & substitution

Various formal techniques and tools have been proposed by several authors to handle system substitution. They use different forms of substitution to describe system adaptation, system reconfiguration or system autonomy.

2.3.1 System reconfiguration

First, many formal tools are used to ensure the correctness of dynamic system substitution in general. In [Bha13], π-calculus and process algebra are used to model systems and exploit behavioral matching based on bi-simulation to reconfigure system appropriately. An extended transaction model is presented to ensure consistency during reconfiguration of distributed systems in [PLB01].

The B method is applied for validating dynamic system substitution of component-based distributed systems using proof techniques for consistency checking and model-checking for timing requirements [LDK11]. A high-level language is used to model architectures (with categorical diagrams) and to operate changes over a configuration (with algebraic graph rewriting) [WLF01].

2.3.2 Fault tolerance

Second, system substitution has been defined to ensure system dependability. Dynamic system substitution can be seen as part of a fault-tolerance mechanism which represents a major concern for designing dependable systems [LCR06; LR14]. Rodrigues *et al.* [Rod+12] presented the dynamic membership mechanism as a key element of a reliable distributed storage system. Event-B is demonstrated in the specification of cooperative error recovery and dynamic reconfiguration for enabling the design of a fault-tolerant multi-agent system, and to develop dynamically reconfigurable systems to avoid redundancy [PTL12; PTL13; Tar+12]. Moreover, this approach enables the discovery of possible reconfiguration alternatives which are evaluated through probabilistic verification.

2.3.3 Autonomic computing and self-$\star$ systems

Third, dynamic system substitution is used to meet several objectives of autonomic computing [PH05; An+15] and self-adaptive systems [Wey+12; Lem+13] such as self-configuration and self-healing. The self-configuring systems require dynamic reconfiguration that allows the systems to adapt automatically to changes in the environment. Similarly, the dynamic reconfiguration makes it possible to correct faults in self-healing systems. Note that we have identified some approaches dealing with adaptive systems that address non-functional requirements [FGT12; Pot13; MPS14].

Next steps In our case, we address system substitution in two situations. The first case is discrete systems. It will be detailed in Chapter 5 and illustrated with the modeling of web services compensation. The second case is hybrid systems. It will be presented in Chapter 7 and illustrated with a controller for a cyber-physical system. In both cases, we will use Event-B to model the systems.

3 Use cases

In this chapter, we introduce two use cases for our study of system substitution: a discrete system and a continuous system. For both cases, we give the particularities, define the requirements for the case study and overview the existing formal approaches used to address them in the state of the art.

3.1 Discrete case: e-commerce web services

3.1.1 Web services: Introduction

The important increase of the use of the web led to the availability of a huge amount of web services. These services can be triggered through web browsers or web applications. The need to compose such services to build more complex services appeared thereafter. The offered composition mechanism led to the emergence of a new programming paradigm. Languages and notations to define services compositions like BPMN [OMG14], XPDL [Wor08], or BPEL [OAS07] have been designed. They offer different features to compose basic and/or composed web services. Several composition operators are embedded in these languages, leading to the design of complex web services compositions.

Similar to the usual complex systems, web service compositions may exhibit inappropriate behaviors in the presence of failures. Therefore, the above languages have been equipped with compensation mechanisms to express running services recovery in case of failures. Compensation is defined as a suspension of the currently running process or activity and a transfer of the execution to a compensating process or activity. For example, BPEL defines a *compensate* operator to compensate an activity defined in a *scope* by another activity when an error is detected. The modalities of the compensation are chosen at design time. The semantics of this mechanism is given informally by the standard.

The lack of formal semantics and of theoretical foundations has been identified in the available definitions of these mechanisms in the standards describing these languages. Indeed, the defined mechanisms do not ensure safety of the compensation, which represents a major concern in particular in the case of transactional web services. In most of the defined languages, ensuring compensation correctness is left to the designer and there is no guarantee that the compensation is correct. Checking that the compensating activity equivalently repairs, degrades or upgrades the compensated activity would help the designers in defining their compensation handlers.

This will be studied in Chapter 5.

3.1.2 Modeling web services compensation

Formal methods have proved their usefulness in the design of correct systems. Several formal approaches for modeling and analyzing web services compositions and languages have been proposed [BBG07]. They promote the use of mathematical foundations to analyze web services compositions. Compensation has been studied from the behavioral point of view and only limited attention has been paid to the functional correctness of the repair due to the limitation of the set up formal methods. All these approaches mention the lack of formal semantics in traditional web services composition and workflow standardized languages like BPEL or BPMN.

When analyzing the state of the art, one can identify three categories of formal methods studying the topic of formal modeling and verification of web services compositions.

In [LM07], the authors give a formalization of the composition operators of the BPEL language using the π-calculus. This work shows, with a simple set of operators, how the whole BPEL language is formalized. Petri nets were used by [HSS05; Loh+08; Aal+09] to encode BPEL constructs and check classical Petri nets properties like deadlock or workflow termination.

Classical state-transition systems have been set up by [Fos+06; Nak06; He+08; MP09] to formalize web services compositions and compatibility problems. Model checking techniques were used to check the correctness of the defined behaviors.

Process algebra based techniques also addressed the problem of web services compositions. The LOTOS algebra was studied by [SBS04] and [Fer04]. The CADP model checker was set up to check the correctness of the described compositions. Butler *et al.* proposed operational or trace semantics for long-running business transactions using CSP [BHF05] or variants of CSP with support for compensation (StAC [BF04] and Compensating CSP [BR05]). The semantics of compensation, specified using a set of primitives, are also studied in [Bru+05]. These approaches have extensively used abstraction techniques, mainly abstracting data, in order to avoid the state number explosion problem due to the state space exploration used by these techniques. As a consequence, they have mainly addressed behavioral aspects thus neglecting the functional correctness. However, the data aspects of transactions were modeled using the B notation in [BFN05].

The third category of approaches relates to the refinement and proof-based techniques. Here we can mention the use of two state-based formal methods that exploit refinement: the ASM (Abstract State Machines) method for modeling by refinement BPMN workflows [BT08] and the Event-B method [AA09; AA10; BW10; AA13]. In both methods, the functional and behavioral aspects have been addressed, and the Event-B based approach proposed to encode, by refinement, the web services decomposition mechanism available in BPEL. This approach will be leveraged in this thesis.

The previously mentioned approaches proposed formal models and verification techniques for services compositions operators available in languages like BPEL and BPMN. In all the previous approaches, a clean semantics has been defined and several properties related to deadlock, termination, correct behavior, *etc.* have been verified, either by model checking or by proof-based approaches.

In the same way, the developed approaches have studied various kinds of compensation. Indeed, we can mention dynamic reconfiguration mechanism studied by [Abo+13] with the π-calculus, dynamic adaptation of web services compositions with Petri nets addressed by [LZ13] and [MGZ14], process algebra [Fer04], Self-Healing described by [Ehr+10] and a model for handling transactions with Event-B defined by [AA13]. These approaches introduce error monitors and trigger a defined compensating service.

The previous approaches addressing compensation studied the occurrence of a condition (error, exception, *etc.*) that causes the compensation. As outlined above, they have addressed the behavioral correctness, whatever is the function achieved by the compensating service. In other words, the correctness of the compensation from the functional point of view is not addressed. This is not surprising when analyzing the mechanisms provided by the traditional services composition languages like BPEL or BPMN.

Our objective As mentioned in the introduction, our objective is to go beyond the capabilities of these languages. Our proposal is twofold. On the one hand, it proposes to check the preservation of the functionality of compensation services, and on the other hand, it supports dynamic compensation at runtime. This proposal is close to the approaches dealing with dynamic system reconfiguration.

In our work, we claim that the capability to handle the functional correctness of the compensating service can be addressed as well. We propose to improve the approach based on the Event-B method and defined in [AA13], that we recall in Section 3.1.3, by adding functional correctness conditions so as the compensating service fulfills some relevant functional correctness conditions expressed by invariants. Refinement will be used to preserve such invariant by the compensating service. Our approach integrates results from formal services compositions modeling and verification, and from dynamic system reconfiguration.

3.1.3 Modeling web services composition with Event-B

This section presents an overview of the work achieved to model BPEL web services compositions. The Event-B method has been used to provide formal models of web services compositions. This work addressed different facets of the formalization of web services compositions and a tool was designed to support the defined development process. More precisely, in [AA09], the authors used the Event-B method to model the whole BPEL language constructs and all the services composition operators:

- Event-B contexts and machines have been used to model these constructs. Indeed, functions, types, triggered services, messages, *etc.* have been modeled in an Event-B context. They represent the static definitions of a BPEL definition.

- Then, the dynamic part of a service composition has been defined in an Event-B machine, importing (using the `Sees` clause) the previously described *context* where the basic services are defined. BPEL variables are declared in the `Variables` clause, they define the states of the state-transition system associated to the described BPEL model. The services composition operators defined in the BPEL language like *flow*, *sequence*, *throw*, *etc.* have been formalized by Event-B *events* occurring in the `Events` Event-B clause. These events were synchronized accordingly with the semantics of each BPEL composition operator. The interleaving semantics offered by the Event-B method was used to formalize the different notions of sequential, parallel, choice and iteration compositions.

The proposed approach proved useful to formalize BPEL web services compositions defined in a single definition. Several relevant properties have been proved: message loss, no call with empty message, no deadlock, functional properties, *etc.* have been expressed in the obtained Event-B machine and proved using the prover associated to the Rodin Platform.

As a second step, [AA10] addressed the web services compositions development process. Decomposition of high level BPEL web services compositions has been studied by exploiting Event-B refinement. The decomposition operator defined in BPEL, has been encoded by a refinement operation in [AA10]. This mechanism offers a stepwise development of web services compositions. The defined mechanism allows the developer to introduce gradually the properties to be fulfilled by the defined services compositions. The whole approach has been described in [AA13].

Finally, in [AA15], transactions have been addressed. The *compensate* BPEL operator characterizing the compensation of a service defined within a *scope* has been formalized. A set of Event-B events supporting the transfer of control from one service to another one has been defined. This transfer is parameterized by an invariant that defines the properties of this compensation, but no specific requirements is set on this invariant. The properties verified in this work were the absence of invocation with empty message, deadlock freeness, reachability of a given state and particularly the terminating state and basic transactional properties related to the triggering of the compensating service.

But, as mentioned above, the defined approach of [AA15] addresses compensation from a behavioral point of view like in the approaches of the literature. Indeed, this approach does not handle the functional correctness of the compensation since conditions on the invariant are not explicitly set. The approach only checks that if a compensation is triggered, then it becomes effective. In order to address this problem, we have sketched in Chapter 5 the first step towards a formal Event-B method that ensures correct service compensation in the case of equivalence. This approach to handle compensation correctness has been generalized in Chapter 8 at a meta level, still using Event-B, in order to guarantee that the methodology for service compensation works for any web services composition.

3.1.4 Web services: Case study

The case study used to illustrate our approach is a simple scenario borrowed from electronic commerce. We consider a simple web application enabling the purchase of a set of products from a supplier. This composition describes a sequence of actions performed by a user. He or she

- selects some products in a cart,
- pays the corresponding total amount of money,
- receives an invoice from the purchasing system,
- then the products are delivered by the logistics part of the system.

This sequence of events is depicted by a simple state transition system in Figure 3.1. The application can be described as a composition (a sequence) of web services corresponding to the labels *Selection*, *Payment*, *Invoicing* and *Delivery* of this state-transition system.

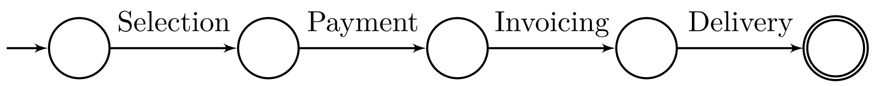


Figure 3.1 – A simple state-transition system describing a sequence of services for purchasing products

To address the compensation problem, we consider that a compensation condition occurs during the selection of the products. We suppose that during the *selection* activity, a failure occurs due to an error on the supplier website. At this step, the system triggers a compensating service. The compensation is composed of two services running in parallel. Each of these services fills a cart of products so that the purchase can be pursued. When the selection is completed, the union of these two carts must contain the set of products expected by the user.

The main requirements for compensation are stated as follows.

- **Correct compensation.** *The compensation shall ensure that the user has purchased the expected set of products whether the products have been purchased* *from one single website with one cart or from two different websites with two carts.* This requirement advocates to take care of the definition of correct compensating services.

- **Compensation at runtime.** *The set of products already available in the cart when a failure occurs shall be preserved by the compensation.* This requirement leads to the definition of a process restoring the state of the halted service.

This case study illustrates several compensation scenarios. We will show in Chapter 5 how the compensation can be formally verified and how different scenarios of equivalent, degraded or upgraded compensations are possible in the proposed approach supported by the Event-B method.

3.2 Continuous case: hybrid systems

System substitution may be instantaneous when state restoration consists in restoring state variables that fulfill the specification invariant. The case of web services compensation mentioned above and studied in Chapter 5 is an instantaneous system substitution. But, in case of hybrid systems, substitution may take some time. This section addresses the case of system substitution where the substitution process needs a certain amount of time. Thus, we must preserve a "safe" behavior of the system during the substitution time.

3.2.1 Hybrid systems: Introduction

According to Lee [LS14], cyber-physical systems (CPS) [LS14; Lee14; Lee15; Akk+16] are defined as *integrations of computation, networking, and physical processes. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa*. The software (the controller) interacts with the physical environment (the plant) in a closed-loop scheme where input from sensors are processed by the controller that generates outputs to the actuators. Moreover, the physical plants are characterized by continuous behaviors while the software controller relies on discrete computations. Internet of Things (IoT), Industrial Internet, Smart Cities, Smart Grid, Smart systems (*e.g.*, cars, buildings, homes, manufacturing, hospitals, appliances), transportation systems, medical devices, . . . are some of the application domains in which CPS take part. Nowadays, *one challenge* is to design trustworthy CPS. The development of safe CPS software controllers using rigorous and formal modeling techniques contributes to reach this challenge.

A key characteristic of CPS is their sensibility to changes which may occur in case of failure, loss of quality of service, maintenance, etc. These changes must be handled by these systems and the service offered by these systems must be preserved as much as possible. Autonomy, adaptation, reconfiguration are some of the requirements associated to CPS design requirements when changes occur. It can be used to ensure high availability in case of failure as required for safety critical systems such as avionics, nuclear, automotive and medical devices, where failure could result in loss of lives, as well as reputation and economical damages. It is important to maintain the running state of a given system in case of any failure by preserving the required behavior in the recovering substitute system. So, *another challenge* in the design of CPS relates to handling changes while preserving the safe behavior of the CPS, or offering upgraded or degraded behaviors.

We claim that formal methods are good candidates to handle these challenges. We address the development of trustworthy CPS. In particular, we contribute to fulfill two main requirements associated to the two previously identified challenges.

- *Modeling both continuous and discrete behaviors.* The software component *controls* the interaction that shall be *soundly* designed from the physical plant described by laws issued from physics (mechanical, electricity, ...). The main questions are related to the use of discrete models by the software while the physical plant is modeled by continuous functions over continuous time (solutions of differential equations) and to the semantic relation between discrete and continuous models. The software (or controller) should have a *correct view* of the *continuous behaviors* and these issues require mathematical foundations as well as foundations for system engineering. The CPS software implements a discretization of these functions in order to control the CPS plant. Proving the correctness of discrete implementations of continuous controllers is a key challenge in the CPS correctness proof. Formal methods play an important role in verifying the system requirements to check the correctness of functional requirements, including the required safety properties. Chapter 6 studies the formal modeling of continuous behaviors.

- *Handling reaction to changes.* Another key requirement for the design of trustworthy cyber-physical systems is the capability of a system to react to changes (*e.g.*, failures, quality of service change, context evolution, maintenance, resilience, *etc.*). The development of such systems needs to handle explicitly, and at design time, the reactions to changes occurring at runtime. Indeed, to prevent a system failure, controllers must react according to environment changes to keep a desired state or to meet minimum requirements that maintain a safety envelope for the system. Mostly, safety critical systems use reconfiguration or substitution mechanisms to prevent any (random) failure, or losing the quality of system services required for system stability. Hybrid system substitution is studied in Chapter 7.

3.2.2 Hybrid systems & formal methods

The development of techniques and tools to handle the correct design of cyber-physical systems has attracted many researchers. Traditional approaches are based on a formal mathematical expression of the problem using real numbers to model continuous time and differential equations to express the behavior model of the studied hybrid system. Then this model is simulated within simulation techniques in order to check its properties. Ptolemy [Pto14] is a good representative of such an approach.

In the past years, several approaches, relying on formal methods, for the development of trustworthy cyber-physical systems have been proposed. They may be gathered in two categories: model checking-based approaches and proof-based approaches.

Model checking and bounded model checking

According to the nature of the handled differential equations, different approaches have been proposed.

When a hybrid system is described by linear or affine differential equations, then model checking [CGP99] techniques can be applied. Hybrid automata [Alu+95; Hen00] are used to model such systems. Tools like HyTech [HHW97], d/dt [ADM02], PHaVer [Fre08] or SpaceEx [Fre+11] have been developed to handle the specification of these systems. They perform exhaustive search and they have proved successful to establish properties like reachability.

Nonlinear hybrid systems support the description of a richer dynamics of the studied systems than linear ones. But, in this case and since reachability for nonlinear systems is not decidable, these approaches do not guarantee termination. So, the benefits of the above mentioned tools resides more in the analysis of the counterexamples they produce rather than on the verification capabilities they offer.

In the case of nonlinear hybrid systems, numerical methods are used when specific assumptions on the boundedness of the continuous variables (bounded horizon) are set. Tools like Flow* [CÁS13] or iSAT [Frä+07] and iSAT-ODE [Egg+11] and dReal/dReach [GKC13b; GKC13a; Kon+15] use bounded model checking for reachability analysis.

All the previous approaches use model checking and suffer from the classical problems encountered by model checking related to state space explosion and to the boundedness of the considered variables. However, these techniques enable automatic verification which is crucial for industrial applications. In order to tackle these limits, classes of automata can be studied through logical analysis [IMN13].

Proof-based approaches

Another category of formal techniques addressing formal modeling of hybrid systems is based on proof techniques and symbolic verification. These approaches support the description of any category of hybrid systems and offer semi-automated tools to handle unbounded variables (*i.e.* unbounded horizon). Axiomatization of the real numbers theory and of the theory of control for linear or nonlinear differential equations is a pre-requisite for the use of these approaches.

Our work belongs to this category of techniques.

S. Boldo *et al.* approach with Coq and Coquelicot In [Bol+14] the authors use the one-dimensional acoustic wave equation case study to illustrate their approach. A program (in the C programming language), encoding a discrete representation of the continuous differential equation describing the behavior of this case study, is annotated using two distinct sets of annotations: one relates to the continuous definitions (derivation, approximation with Taylor series *etc.*) and the second deals with discrete aspects of the program (loop invariants, pre-conditions and post-conditions of the used functions, *etc.*). These annotations complete and enrich the controller description with descriptions of the plant behavior. They are used to prove the stability and convergence of the programmed numeric scheme solving the differential equation. The Frama-C[1]/Jessie [Mar07]/Why [FM07] tool suite generates proof obligations. They are proved either automatically or interactively using SMT solvers, Gappa[2] or interactively using the Coquelicot [BLM15] Coq [BC04] library.

Finally, note that the developed approach also deals with floating-point arithmetic manipulated by the analyzed C program.

A. Platzer approach and KeYmaera tool In [Pla08], A. Platzer defines hybrid programs to describe continuous and discrete behaviors of hybrid systems in a closed-loop modeling approach together with a logic and its proof system, namely dynamic logic for dynamic systems. These programs give an abstract description of a hybrid system. Discrete and continuous behaviors are described as hybrid programs using discrete assignments, continuous variables evolution along differential equations, non deterministic choices, iteration, *etc.*

Properties on the defined hybrid programs are expressed within the dynamic logic constructs offering classical first order logic constructs together with the $\Box$ (denoted $[\cdot]$) and $\Diamond$ (denoted $\langle\cdot\rangle$) modalities to express invariants and reachability properties. KeYmaera [Que+16] is the semi-automatic prover tool supporting the proof process for the defined hybrid programs. It supports the defined dynamic logic proof system. The approach has been applied to model hybrid systems like car control system [Que+16], train control system [PQ09] and flight collision avoidance system [PC09].

Compared to Event-B-based approaches detailed below, it does not provide a built-in refinement development operator.

J.-R. Abrial and W. Su approach with Event-B The work initiated in [SAZ14] proposes to model first the discrete events of a hybrid system and then refine each event by introducing the continuous elements. Events are partitioned into *environment* events and *control* events. It includes the use of a “now” variable and a “click” event that jumps in time to the next instant where an event can be triggered. The authors do not study the possible definition of the continuous parts by means of differential equations. Only arithmetic on emulated reals is used. In [SA14] the authors enrich the work of [SAZ14] by incorporating analytical results from the study of differential equations into the Event-B models through the complementary use of Matlab/Simulink.

M. Butler, J.-R. Abrial and R. Banach approach with Event-B The authors of [BAB16] extend the approach of [SAZ14] using the Theory plug-in to

[1] http://www.frama-c.cea.fr/

[2] http://gappa.gforge.inria.fr/

define a theory of real arithmetic (see Section 1.8).

In this approach, hybrid systems are expressed as continuous evolutions of variable values over time. These evolutions follow monotonic functions ensuring that no bad behavior occurs between two observed discrete steps. The approach consists in defining first the continuous behavior. It is first refined by introducing modes. Then a second refinement introduces a control strategy defining discrete control steps. Finally, a last refinement merges (*i.e.* eliminates) the continuous variables. This refinement describes the final controller, it contains discrete steps only. The approach has been illustrated by the design of a controller for a water tank.

R. Banach approach with Hybrid Event-B The second proposed approach based on Event-B, initiated by Banach, is Hybrid Event-B [Ban+15]. This is an extension of Event-B which includes *pliant* events [Ban13] (as opposed to *discrete* events) as a way to model continuous behavior, allowing the direct use of differential equations in the modeling. However, there is no tool currently supporting this extension whereas our approach enabled us to develop and prove the models using available tools. Banach also worked on similar topics with ASM [Ban+11; Ban+12]. Applications of the approach have been proposed in [Ban+14; Ban16a; Ban16b].

Modeling of time All the proof-based approaches summarized above use theories of reals. These theories support the definition of relevant properties like continuity of functions or invariants to characterize real variables regions or to describe Taylor series. The approaches of Platzer [Pla08; Que+16], Banach [Ban+15] and Boldo [Bol+14] support the explicit definition of differential equations. Time is implicitly considered in these approaches through these differential equations. [Bol+14] deals with C programs using a suite of proof tools while KeYmaera [Que+16] is deployed on hybrid programs that provide an abstract model of a hybrid system in a closed-loop modeling approach. Observe that there are no bibliographic references between the approaches of [Bol+14] and of [Que+16]. In [Ban+15], the adopted approach is similar to [Pla08]. The added value of this approach is the use of refinement to define a stepwise formal development preserving the invariants in the different refinement levels. But, up to now, there is no tool supporting the approach.

The approaches of [SAZ14] and [BAB16] use Event-B and the Rodin Platform [Abr+10] to model hybrid systems in a closed-loop model. Time is explicitly modeled using a specific state variable. The authors consider continuous functions and they define discrete and continuous transitions preserving invariants characterizing the correct behavior of the described hybrid system. Refinement proved useful for the stepwise design of a hybrid system. The approach is tool-supported, all the developments following these approaches can be formalized within Rodin.

3.2.3 Hybrid systems: Case study

Hybrid systems

The description of the behavior of hybrid systems relies on the definition of continuous behavior characterized by continuous functions over time. Figure 3.2 depicts a graphical representation of such functions. To control a system, in particular for system reconfiguration, it is required to observe (the feedback behavior of the function) and to control (keep or change system mode) the system. Such observation and control are performed by a software requiring the discretization of continuous functions. When software is used to implement such controllers, time is observed according to specific clocks and frequencies. Therefore, it is mandatory to define a correct discretization of time that preserves the observed continuous behavior introduced previously. This preservation entails the introduction of other requirements on the defined continuous function. Note that, in practice, these requirements (assumptions) are usually provided by the physical plant.

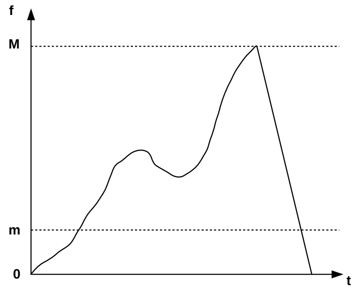


Figure 3.2 – Example of the evolution of the function f

Table 3.1 – Requirements in the abstract specification.

At any time, the feedback information value of the controlled system shall be less or equal to M in any mode.	Req.1
At any time, the feedback information value of the controlled system shall belong to an interval $[m, M]$ in *progress* mode.	Req.2
The system feedback information value can be produced either by f, g or $f + g$ (f and g being associated to Sys_f and Sys_g)	Req.3
The system Sys_f may have feedback information values outside $[m, M]$	Req.4
At any time, in the *progress* mode, when using Sys_f, if the feedback information value of the controlled system equals to m or to M, Sys_f must be stopped.	Req.5

Substitution

We consider two continuous functions f and g characterizing the behavior of two hybrid systems Sys_f and Sys_g. We also assume that these two systems maintain their feedback information value in the safety envelope $[m, M]$. As a consequence, these two systems substitute each other since they fulfill the same safety requirement. In this chapter, the studied scenario consists in substituting Sys_f by Sys_g after a failure occurrence (see requirements of Table 3.1).

Figure 3.3 shows the substitution scenario in both continuous and discrete cases. The X axis describes time change and the vertical dashed lines model state transitions. Observe that during the repairing process function f (associated with Sys_f) decreases due to its failure while function g (associated with Sys_g) is booting.

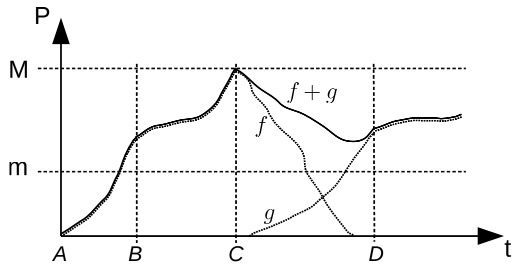


Figure 3.3 – Example of the evolution of the functions f, g and $f+g$

In our approach, we use refinement to fulfill the first requirement. Several refinements may implement a single specification. They characterize a class of systems that are candidate for substitution. Regarding the second requirement, a relation restoring the state variables of the substituted and substitute system is defined. It shall preserve the invariant and properties of the original specification.

In the next part, we will start by introducing a general substitution model in Chapter 4. Then, the discrete case will be presented in Chapter 5 and the continuous case in Chapter 7, after having studied the modeling of continuous systems in Chapter 6.

Part II

Contributions

We rely on formal methods, more precisely the Event-B formal method [Abr10] that provides proof and refinement for state-based models, to describe both the studied systems and the introduced substitution operation. We have chosen to describe systems as state-transition systems.

In this part dedicated to the contributions provided in this thesis, we first define in Chapter 4 a generic substitution model while explaining how it supports the expression of different substitution characteristics and how it relates to proof obligations. Then, in Chapter 5, we show how our proposal applies to discrete system substitution. The case of service compensation is shown as an illustrating example. Chapter 6 presents how hybrid systems can be modeled and verified in Event-B by going from continuous models to discrete ones using refinement. We are then able to model substitution occurring in hybrid systems in Chapter 7. Finally, Chapter 8 presents a generalization of our generic model, that enables us to define a common core part of the proofs. It also shows a model that can be refined to specific systems. Again, the case of service compensation is shown as a particular system captured by this generic model.

4 A generic substitution model

Chapter organization. The main contribution of this chapter is presented in Section 4.2. It describes a stepwise methodology for the design of a correct system substitution operation. Proof obligations derived from the defined operation are presented in Section 4.3. The possible ways of applying the defined operation are discussed in Section 4.4. Finally, a conclusion summarizes our contribution in the last section.

4.1 Introduction

Our work aims at defining a *generic* correct-by-construction approach to model system substitution at runtime.

Objective of this chapter. We want to model system substitutions and prove the correctness of these substitutions. That is why we define a generic framework able to model substitutions with various characteristics while being able to prove the correctness of these substitutions through the identification of the related proof obligations.

4.2 System substitution

The availability of several refinements for a given specification means that several systems may implement a single specification. Each of these systems behaves like the defined specification. The systems that refine the same specification can be gathered into a class of systems. The availability of such a class makes it possible to address the problem of system substitution or system reconfiguration. The stepwise methodology for system substitution that we propose, considers one system of this class as a running system, and substitutes it by another system belonging to the same class. Indeed, when a running system is halted (in case of failure or loss of quality of service, *etc.*), a system of this class can be chosen as a substitute. In this chapter, we describe a formal methodology allowing system developers to define correct-by-construction system substitution or system reconfiguration. By “correct”, we mean the preservation of safety properties expressed by the invariants.

4.2.1 A stepwise methodology

Our approach to define a correct system substitution setting is given in several steps. This stepwise methodology leads to the definition of a system substitution operator whose properties are discussed later.

- **Step 1.** *Define a system specification.* A state transition system characterizing the functionalities and the suited behavior of the specification system is defined.

- **Step 2.** *Characterize candidate substitute systems.* All the refinements of the specification represent substitutes of the specified system. They preserve the invariants properties expressed at the specification level. A class of substitutes is obtained. It contains all the systems refining the same specification.

- **Step 3.** *Introduce system modes.* Modes are introduced to identify which system is running *i.e.*, those that have been halted and the remaining available systems for substitution. A mode is associated with each system, and at most one system is running.

- **Step 4.** *Define system substitution as a composition operator.* When a running system is halted, the selected substitute system becomes the new running system. During this substitution, the state of the halted system shall be restored in the substitute system. Restoring the state of the halted system consists in copying the values of the state variables of the halted system to the variables of the state of the substitute system. To formalize this operation, a sequence of two specific events is introduced. The first event, named `fail`, consists in halting the running system and switching it to a failure mode.

The second one, namely `repair`, restores the system state and switches the control to the substitute system. Because `repair` depends on the modeling of the internal state of both systems, it has to be explicitly defined for each pair of systems (it is a parameter of the substitution operator). Here, we consider only pairs of systems where the relation between the internal state of the halted system and of the substituted system can be explicitly defined.

4.2.2 An Event-B model for system substitution

In this section, we give an overview of the Event-B models corresponding to the stepwise methodology presented above. First a specification *Spec* of an abstract system is given, then we show how a source system S_S defined as a refinement *SysS* of the machine *Spec* can be substituted by a target system S_T defined as a refinement *SysT* of the same machine *Spec*. Two events `fail` and `repair` for halting a system S_S and for transferring the control to the target system S_T are introduced.

Step 1. Define a system specification

The specification of the system is given by an abstract description of its functionalities and its behavior. An Event-B machine *Spec*, corresponding to the one in Model 2.2 page 26, defines the system specification. In that model, the behavior is defined by a single event, but there is no explicit limitation on the number of events.

More events may be introduced to define this behavior, we have just limited our description to one single event.

Step 2. Characterize candidate substitute systems

As stated above in Section 4.2.1, a class of substitute systems is defined as the set of the systems that are described as an Event-B refinement of the original Event-B machine *Spec*. Two systems *SysS* and *SysT* described by the Event-B refinements in Models 4.1 and 4.2 are substitute systems for the system described by the specification *Spec*. Note that several refinement steps may be required before the final models of the substitute systems are obtained.

On these two refinements *SysS* and *SysT*, we note the presence of:

- new sets of variables,
- an invariant describing the properties of the system and gluing the variables with the ones of the abstraction in the *Spec* machine,
- new events that may be either added or refined in order to describe the behavior of the new variables or define behaviors that were hidden in the specification,
- a variant: an expression whose value strictly decreases and which models the progress (or position) of the system, while guaranteeing its termination.

```
Machine SysS
Refines Spec
Sees C0
Variables v_S
Invariants I_S(s, c, v_S, v_A)
Variant VN_S
Events
  Event Initialisation ≙
    Begin
      v_S :| D_S(s, c, v'_S)
      VN_S :| VN_S_InitValue
    End
  Event s_evt ≙
    Any x_S
    Where
      G_S(x_S, s, c, v_S)
    Then
      v_S :| BA_S(x_S, s, c, v_S, v'_S)
    End
    ...
  Event Evt Refines Evt ≙...
End
```

Model 4.1 – Machine *SysS* (reminder)

```
Machine SysT
Refines Spec
Sees C0
Variables v_T
Invariants I_T(s, c, v_T, v_A)
Variant VN_T
Events
  Event Initialisation ≙
    Begin
      v_T :| D_T(s, c, v'_T)
      VN_T :| VN_T_InitValue
    End
  Event t_evt ≙
    Any x_T
    Where
      G_T(x_T, s, c, v_T)
    Then
      v_T :| BA_T(x_T, s, c, v_T, v'_T)
    End
    ...
  Event Evt Refines Evt ≙...
End
```

Model 4.2 – Machine *SysT*

We consider that both *SysS* and *SysT* see the context *C0* of the specification *Spec*, and we assume that no new specific element is needed for their own contexts.

Step 3. Introduce system modes

The introduction of modes is a simple operation consisting in defining a new variable m (standing for *mode*). The values of the mode variable may be either the system identifier (S or T) or the value F to represent a halted system in a failure mode. Moreover, the invariant related to each substitute system shall be valid when the variable m is equal to that system identifier. Models 4.3 and 4.4 show the description of the systems S and T with introduced mode. Again, each of the machines *SysS** and *SysT** refine the original specification *Spec*. At this step, we also anticipate any name clashes by renaming some elements through the addition of a prefix.

Step 4. Define system substitution as a composition operator

The machines *SysS** and *SysT** are composed into a single Event-B machine with two new events `fail` and `repair`. The role of the substitution operation is to enable the following sequence of events.

1. The source system S is the first running system. The variable mode m is

Machine SysS$*$
Refines SysS
Sees C0
Variables v_S, m
Invariants $m = S \Rightarrow I_S(s, c, v_S, v_A)$
Variant VN_S
Events
  Event Initialisation $\widehat{=}$
    Begin
      $m := S$
      $v_S :\mid D_S(s, c, v'_S)$
      $VN_S :\mid VN_S\_InitValue$
    End
  Event s_evt $\widehat{=}$
    Any y_S
    Where
      $m = S \wedge G_S(y_S, s, c, v_S)$
    With
      $y_S = x_S$
    Then
      $v_S :\mid BA_S(y_S, s, c, v_S, v'_S)$
    End
    ...
  Event Evt **Refines** Evt $\widehat{=}$...
End

Model 4.3 – Machine *SysS**

Machine SysT$*$
Refines SysT
Sees C0
Variables v_T, m
Invariants $m = T \Rightarrow I_T(s, c, v_T, v_A)$
Variant VN_T
Events
  Event Initialisation $\widehat{=}$
    Begin
      $m := T$
      $v_T :\mid D_T(s, c, v'_T)$
      $VN_T :\mid VN_T\_InitValue$
    End
  Event t_evt $\widehat{=}$
    Any y_T
    Where
      $m = T \wedge G_T(y_T, s, c, v_T)$
    With
      $y_T = x_T$
    Then
      $v_T :\mid BA_T(y_T, s, c, v_T, v'_T)$
    End
    ...
  Event Evt **Refines** Evt $\widehat{=}$...
End

Model 4.4 – Machine *SysT**

initialized to the value S in order to transfer the control to the events of the system S.

2. When a halting event occurs, the `fail` event is triggered. This event changes the value of the mode variable m to the value F. At this state, the system S is stopped and the invariant I_S is valid at that current state. Note that the event `fail` can be triggered for any reason in the current formalization.

3. At this stage, the `repair` event is triggered because its guard ($m = F$) is enabled (Model 4.6). This event serves two purposes. On the one hand, it restores the state of the halted system by defining the values of the variables v_T of the substitute system S_T and on the other hand, it sets up the variable VN_T used to express the variant, to allow the restart of the system S_T at the suited state (or the closer state). Finally, the mode is changed to T so that the control is transferred to the substitute system S_T.

   The definition of the `repair` event (Model 4.6) implies the definition of state restoration. The new values of the variables of system S_T must fulfill safety

```
Event fail ≙
  Where
    m = S
  Then
    m := F
  End
```

Model 4.5 – Extract of event `fail`

```
Event repair ≙
  Where
    m = F
  Then
    // New values for state variables
    v_S, v_T := ...
    // New values for variants
    VN_T := ...
    // Change mode
    m := T
  End
```

Model 4.6 – Skeleton of event `repair`

conditions in order to move the control to S_T in order for the invariant I_T to hold in the recovery state. In other words, specific proof obligations are associated to the `repair` event.

4.2.3 Substitution as a composition operator

As stated above, the `repair` event shall be defined so that the state restoration preserves the safety properties described in the invariants. The definition of this event is completed in Model 4.7.

At this level, two predicates are defined.

1. The *Recover* predicate characterizes the new values of the variables v_T such that the invariant I_T holds in the next state. It represents the *horizontal* invariant that glues the state variables of system S_S with the variables of system S_T.

2. The *Next* predicate describes the next value of the variant. It determines, which state in the system S_T, is used as the new restoring state preserving the invariant I_T.

```
Event repair ≙
  Where
    m = F
  Then
    v_S, v_T :| Recover(v_S, v_T, v'_S, v'_T)
    VN_T :| Next(V_S, V'_T)
    m := T
  End
```

Model 4.7 – Extract of event `repair`

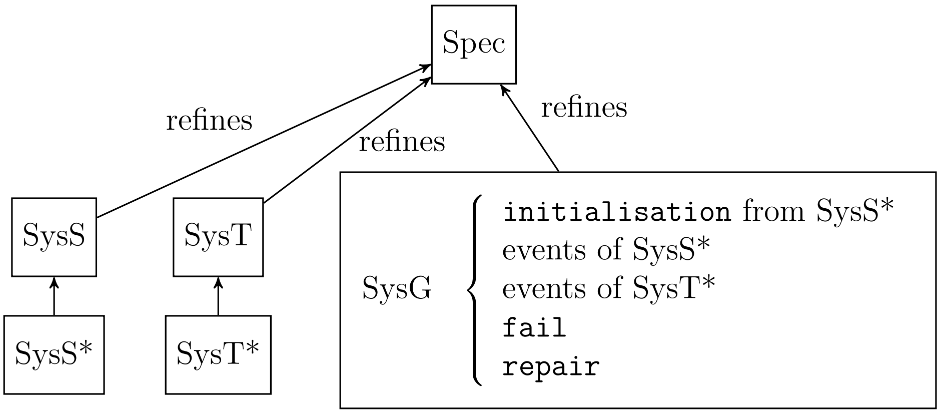


Figure 4.1 – Systems

4.2.4 The obtained composed system with substitution

Once the `fail` and `repair` events have been defined, the obtained model is composed of the two systems S_S and S_T. The sequence described above is encoded using a predetermined sequence of assignments of the mode variable m in the corresponding events.

Moreover, the invariant of the final system is defined by cases depending on the value of the mode variable. When the system S_S is running, the invariant I_S holds, when the system S_T is running, the invariant I_T holds and finally, as stated previously, the invariant I_S holds when the system S_S is halted and being substituted. The obtained invariant is a conjunction of three implications.

The global system is again described as a refinement of the original specification. It is formalized by the Event-B machine *SysG* as shown in Model 4.8.

$$\boxed{S_G = S_S \circ_{(Recover,Next)} S_T \text{ refines } Spec} \tag{4.1}$$

Finally, as defined in Equation (4.1), we can define a composition operator $\circ_{(\ldots,\ldots)}$ parameterized by the *Recover* and *Next* predicates.

The refinement relations are summarized in Figure 4.1.

4.3 Proof obligations for the system substitution operator

The proof obligations resulting from the definition of our substitution operator concern invariant preservation by the different events of the Event-B machine Sys_G. Let us analyze these proof obligations.

- For the initialization and the events of system *SysS*, the preservation of the invariant is straightforward. The proofs are those that have been performed for the refinement introducing modes in the previous step.

```
Machine SysG
Refines Spec
Sees C0
Variables v_S, v_T, m
Invariants    (m = S ⇒ I_S(s, c, v_S))
            ∧ (m = F ⇒ I_S(s, c, v_S))
            ∧ (m = T ⇒ I_T(s, c, v_T))
Variant VN_S + VN_T
Events
  Event Initialisation ≙
    Begin
      m := S
      v_S :| D_S(s, c, v'_S)
      v_T :| ⊤
      VN_S :| VN_S_InitValue
      VN_T :| 0
    End
  Event s_evt ≙
    Any x_S
    Where
      m = S ∧ G_S(x_S, s, c, v_S)
    Then
      v_S :| BA_S(x_S, s, c, v_S, v'_S)
    End
```

```
  Event Evt Refines Evt ≙ ...
  Event fail
    Where
      m = S
    Then
      m := F
    End
  Event repair ≙
    Where
      m = F
    Then
      v_S, v_T :| Recover(v_S, v_T, v'_S, v'_T)
      VN_T :| Next(V_S, V'_T)
      m := T
    End
  Event t_evt ≙
    Any x_T
    Where
      m = T ∧ G_T(x_T, s, c, v_T)
    Then
      v_T :| BA_T(x_T, s, c, v_T, v'_T)
    End
  ...
End
```

Model 4.8 – Machine *SysG*

- The same situation occurs for the events of system *SysT*. Again, the associated proof obligations are those obtained and proved when introducing modes in the previous step.
- The `fail` event preserves the invariant since it does not modify any state variable except the mode. It preserves the invariant I_S with $(m = S \Rightarrow I_S(s, c, v_S)) \wedge (m = F \Rightarrow I_S(s, c, v_S))$.
- Finally, the `repair` event considers that I_S holds before substitution and it must ensure that the invariant I_T holds after substitution.

So, the introduction of the `repair` event entails specific proof obligations that needs to be discharged in order to ensure the correctness of the substitution. The definition of the Recover predicate is the key point to obtain a correct system substitution. The proof obligations associated to the `repair` event consists first in preserving the invariants and second in restoring the correct variant value.

4.3.1 Invariant preservation proof obligation

Invariant preservation for the `repair` event requires to establish that the invariant I_T of system S_T holds in the recovery state. In other words, under the hypotheses given by the axioms $A(s,c)$, the guard $m = F$, the invariant $(m = S \Rightarrow I_S(s,c,v_S)) \wedge (m = T \Rightarrow I_T(s,c,v_T)) \wedge (m = F \Rightarrow I_S(s,c,v_S))$ and the new variable values $Recover(v_S, v_T, v'_S, v'_T) \wedge m' = T$, the invariant $(m' = S \Rightarrow I_S(s,c,v'_S)) \wedge (m' = T \Rightarrow I_T(s,c,v'_T)) \wedge (m' = F \Rightarrow I_S(s,c,v'_S))$ hold for the variables in the next state. The sequent in Equation (4.2) describes this proof obligation.

$$
\begin{array}{l}
\quad A(s,c),\\
\quad (m = S \Rightarrow I_S(s,c,v_S)) \wedge (m = T \Rightarrow I_T(s,c,v_T)) \wedge (m = F \Rightarrow I_S(s,c,v_S)),\\
\quad m = F,\\
\quad Recover(v_S, v_T, v'_S, v'_T) \wedge m' = T\\
\vdash\\
\quad (m' = S \Rightarrow I_S(s,c,v'_S)) \wedge (m' = T \Rightarrow I_T(s,c,v'_T)) \wedge (m' = F \Rightarrow I_S(s,c,v'_S))
\end{array}
\tag{4.2}
$$

After simplification, the previous proof obligation leads to the definition of the final proof obligation of Equation (4.3) associated to invariant preservation.

$$
\boxed{A(s,c) \vdash I_S(s,c,v_S) \wedge Recover(v_S, v_T, v'_S, v'_T) \Rightarrow I_T(s,c,v'_T)} \tag{4.3}
$$

4.3.2 Variant definition proof obligation

The introduction of the new variant value determines the restoring state in the target system S_T. The predicate $Next$ needs to be defined so that the variant $VN_S + VN_T$ of the global system decreases. It is required to establish that $VN'_S + VN'_T < VN_S + VN_T$. The next value of VN'_T determines the restoring state in system S_T. Since the value of the variant VN_S does not change, only the variant VN_T decreases. The associated proof obligation is given by the sequent of Equation 4.4.

$$
\begin{array}{l}
\quad A(s,c),\\
\quad (m = S \Rightarrow I_S(s,c,v_S)) \wedge (m = T \Rightarrow I_T(s,c,v_T)) \wedge (m = F \Rightarrow I_S(s,c,v_S)),\\
\quad m = F,\\
\quad Next(VN_S, VN'_T) \wedge m' = T \wedge VN'_S = VN_S\\
\vdash\\
\quad VN'_S + VN'_T < VN_S + VN_T
\end{array}
\tag{4.4}
$$

After simplification, the previous proof obligation leads to the definition of the final proof obligation of Equation (4.5) associated to variant definition.

$$
\boxed{A(s,c), I_S(s,c,v_S) \vdash Next(VN_S, VN'_T) \wedge VN_S = VN'_S \Rightarrow VN'_T < VN_T} \tag{4.5}
$$

4.3.3 About restored states

As shown on the proof obligations obtained in Equations (4.3) and (4.5), the definition of the *Recover* and *Next* predicates is identified as the fundamental characteristics for the correct substitution operation.

The *Recover* predicate defines the *horizontal invariant*. This invariant defines the properties needed to restore the state variables of the original halted system in the substitute state variables. It also describes the safety property of the substitute system. According to the definition of this predicate, as discussed in Section 4.4, different substitution cases are identified.

Regarding the *Next* predicate, one can note that any value of the variant that decreases the variant VN_T is accepted. For instance, one could set up the variant to the final state of system S_T meaning that the substitution has been done in the final state. The only condition concerns the *Recover* predicate which shall restore the correct values of the variables in this final state.

4.4 Substitution characteristics

4.4.1 Cold and Warm start

In the approach we have sketched in Section 4.2, this characteristic is handled by the correct definition of the *Recover* and *Next* predicates. Indeed, according to the definition of these predicates, the restored state may be either the initial state (in the case of a cold start) or a state constructed from the current state to be as close as possible to the current state from a functional standpoint (in the case of a warm start).

4.4.2 Identical, included or disjoint sets of state variables

In the framework presented in Section 4.2, v_S and v_T represent the set of state variables for the original and substitute systems. According to the properties linking these two sets in the `repair` event using the *Recover* predicate, different substitution cases occur.

- The sets of variables are identical *i.e.* $v_S = v_T$. The effect of the `repair` event is to restore a new state (correct with respect to the given invariants) after substitution.
- The sets of variables are partially shared *i.e.* $v_S \cap v_T \neq \varnothing$.
- The sets of variables are disjoint *i.e.* $v_S \cap v_T = \varnothing$. The `repair` event transfers the control to a completely new substitute system.

4.4.3 Equivalence, Upgrade and Degradation

Within the provided framework three cases can be identified and handled. The substitute system SysT may be equivalent to the original system SysS, upgrade it (enhance it) or degrade it.

As quality of service is out of scope of our framework, the three previous cases can be described with adequate definitions of the *Recover* and *Next* predicates. In fact, the definition of each case relies on the provided invariants to be preserved during substitution *i.e.* by the `repair` event.

Let us assume that there exist two predicates Φ and Ψ ($\Phi \neq False \wedge \Psi \neq False$) such that $I_S \wedge \Phi \iff I_T \wedge \Psi$, then the three identified cases can be expressed.

- **Equivalence** is obtained when $I_S \iff I_T$. It means that the substitute preserves the same invariant properties as the original system since $\Phi \iff True$ and $\Psi \iff True$. The case study presented in Section 3.1.4 illustrates this case. The set of products purchased with the substitute system *SysT* is identical to the original system *SysS*.

- **Upgrade** occurs when $I_S \wedge \Phi \iff I_T$. Here, the substitute system *SysT* offers more functionalities characterized by the invariant part Φ than the original system. Indeed, $I_T \implies I_S$ which means that the substitute system guaranties the properties that the previous did. Additionally, $I_T \implies \Phi$ which specifies that the substitute system also guaranties the new property Φ.

- **Degradation** is dual to upgrade and it occurs when $I_S \iff I_T \wedge \Psi$. Here, the substitute system looses some of the functionalities characterized by the invariant part Ψ of the original system.

4.4.4 Static or dynamic set of substitutes

In the framework presented in the previous section, we have assumed that the set of substitute systems is known and does not change (static). Modes have been introduced to identify the running system and the selected substitute system is known by the `repair` event.

To handle a mechanism where the set of substitutes would be dynamic, an event managing (adding or removing substitutes) a set of modes corresponding to substitute systems (that refine a common specification) must be added, and the `repair` event must select a substitute in this set.

4.5 Conclusion

This chapter addressed the problem of correct system substitution, where systems are described as state-transition systems. It provides a stepwise correct-by-construction approach based on refinement and proof supported by the Event-B method. It has been published in [BAP16a].

This approach relies on two elements:

1. the definition of a class of systems that implement (*i.e.* refine) the same specification

2. a system substitution operator parameterized by a recovery property, namely a *horizontal invariant*. This composition operator combines two or more systems that refine the same specification. It is parameterized by the *substitution* or *repair property* ensuring that the current state (the state where the source system is halted) is correctly restored in the substitute system.

The defined framework for substitution ensures that, when a system is halted (a failure occurs for instance), the state of the source system is correctly restored to the state of the target system. Depending on the definition of the horizontal invariant, the composition operator entails three types of substitution: equivalent, degraded or upgraded substitute systems can be obtained. This will be expanded in Chapter 5.

Two different substitution relationships have been presented. The first one is a static substitution (corresponding to a *cold start*). It relies on refinement to characterize the set of systems that conforms to the same specification. A class of potential implementation systems are thus characterized by refinement. Here when a system is halted, the state is restored to the initial state of the substitute system. The second one addresses the dynamic substitution (substitution at runtime or *warm start*) which uses state restoration by transferring the control to the adequate state in the substitute system.

Furthermore, the `fail` event can be refined in order to introduce failure conditions like loss of quality of service.

This framework for substitution has been applied to the two use cases presented in Chapter 3. Discrete system substitution is detailed in Chapter 5. Continuous system substitution is presented in Chapter 7, using the work of Chapter 6 on the modeling of continuous systems. A formalization of the generic framework presented in this chapter together with an instantiation of this model for a discrete case are presented in Chapter 8.

5 Discrete systems substitution

Chapter organization. The formal modeling and verification of services compositions within Event-B has been discussed in Section 3.1.3. Our view on service compensation is given in Section 5.2, and Section 5.3 describes the stepwise methodology we have proposed to handle such a formal process for services compensations.

The root model corresponding to the global specification of our case study is given in Section 5.4. Then, in Section 5.5 we give the application of this approach on the defined case study where the specific case of equivalent compensation is detailed. Finally, Section 5.6 presents an overview of the two other compensation cases (degraded and upgraded cases). At the end of this chapter, a conclusion summarizes the key contributions and identifies some research directions.

5.1 Introduction

Objective of this chapter. The objective of this chapter is to show how our approach for system substitution applies to discrete system substitution. We have chosen to illustrate such systems for web service compensation. In this chapter, we advocate the use of invariant preservation in order to formally check the correctness of service compensation. We propose a correct-by-construction approach to handle compensation at runtime and we model service compensation as a particular case of system substitution. It can be used as a ground model for runtime service compensation as defined in languages like BPEL. The approach is based on refinement and proof using the Event-B method. Safety of the compensation is guaranteed by invariant preservation corresponding to a liveness property (*leads-to* property). Three compensation cases are addressed: equivalent, degraded and upgraded compensation cases.

5.2 Our view of compensating activities

One of the main requirements of service compensation is consistency. Indeed, compensation shall:

- complete the functional objective of the compensated service. In our case study, described in Section 3.1.4, this statement refers to the *correct compensation* requirement.
- *safely* transfer the control from one service to another one at runtime by preserving, as much as possible, the completed steps of the compensated service. In our case study, this statement refers to the *compensation at runtime* requirement.

The key idea for service compensation, developed in this chapter, is based on invariant preservation. Invariants are defined at the root level to characterize the functional correctness property associated to the defined services composition. The invariants are preserved in further refinements that shall guarantee this preservation. Invariants are associated to each service, they express the property related to the function accomplished by a given service.

During compensation, the preservation of such invariants by the compensating service is required. To preserve these invariants, a relation, fulfilling safety conditions, shall be defined between the compensated service and the compensating one. In other words, the state of the compensated service shall be restored in the compensating service so as the invariant still holds.

In the rest of this chapter, our approach for service compensation is defined. We address the case of compensating a source service by another target service. We consider that such a compensating service is always available, it belongs to the class of services that refine a global specification of a services composition. Therefore, the compensating service is chosen following the refinement criteria. Any service that refines the same global specification is a good candidate for compensation. Quality of services aspects are not addressed here.

5.2.1 Compensation of a service by another one: definition

Our compensation mechanism relies on the following definition. For a given activity supported by a service a, a source service s is compensated by a target service t if and only if the following holds:

1. Activities defined by the services s and t refine the activity defined by the service a using the gluing invariant I_s and I_t respectively guaranteeing that s and t realize the same function as a.
2. There exists a logical relation, defining an invariant, linking (gluing) the states of the source service s and the target service t. It ensures that a repair action or compensation:
    (a) does not violate the refinement of the activity a,
    (b) defines a recovered state in the target service that satisfies the defined invariant and thus ensures the correct refinement of the global specification.

These two conditions shall be guaranteed by each defined compensation mechanism at runtime. Observe, that compensation can be seen as a specific case of system substitution as introduced in Chapter 4.

5.2.2 The role of the invariant

The invariant plays a key role to ensure that, during compensation, the source and target services fulfill the invariant defined in the global specification. This result is ensured by the correct refinement which introduces the gluing invariant. It shows that the source service s can be compensated by the target service t but it does not provide us with information about the recovery state and thus about compensation at runtime.

So, this definition of the invariant is not enough to guarantee correct state restoration. According to the defined methodology, the developer shall exhibit a specific relation between the state of the source service s when halted and the restored state of the target service. This relation defines the so-called *horizontal invariant*. Moreover, modes are used to manage the switching from the halted service to the compensating one. The mode changes ensure atomicity (discrete case) of the compensation since no other service runs during compensation, and thus no state variable is modified.

When such a relationship and horizontal invariant are provided, different compensation cases become possible: degraded, upgraded or equivalent.

5.2.3 Different compensation cases

Let us assume that services s and t correctly refine the specification described by the activity or service a. This means that both s and t are correct implementations of a. As a consequence, s and t belong to the same class of implementation services for a. Moreover, one can formally assert that service t correctly compensates service s.

Since s and t refine the same service specification a, they both define their own gluing invariant I_s and I_t ensuring the correct refinement of a.

At this stage of our development, we are able to define the relationship between the states of each refined service. Indeed, the following logical relation of equivalence can be expressed. It defines the *horizontal invariant* and different compensation cases.

$$I_s \wedge \phi \Longleftrightarrow I_t \wedge \psi$$

Here ϕ ($\neq$ *false*) and ψ ($\neq$ *false*) define logical expressions to link both invariants. So different cases may occur. This relation leads to the four following situations.

1. $\phi = \psi = \mathit{true}$. This situation describes the case where service s is compensated by an *equivalent* service t. The two services accomplish the same goal.

2. $\phi = \mathit{true}$ and $I_t \nvDash \psi$. This situation describes a case where service t *degrades* service s during the compensation. The I_t invariant does not cover the whole functional specification of s. The compensation does not guarantee that the activity performed by s will remain the same in the compensating service because part of the invariant I_s is supported by ψ.

3. $I_s \nvDash \phi$ and $\psi = \mathit{true}$. This situation describes the case where service t *upgrades* service s during the compensation. It guarantees both I_s and other properties expressed by ϕ. It means that t "*does*" more than s but it preserves the functional properties targeted by s.

4. $\phi \neq \mathit{true}$ and $\psi \neq \mathit{true}$. Finally, this case corresponds to an unknown situation where no information about the compensation can be inferred.

Cases 1, 2 and 3 are considered in this chapter. They correspond to the cases identified in Section 4.4.3 of our methodology for system substitution. They correspond to realistic situations. Case 4 is not useful and is not considered in our work.

5.2.4 Different compensation cases: illustration on the defined case study

In the case study defined in Section 3.1.4, let us consider the *selection* activity corresponding to the web service that proceeds to the selection of a set *cart* of purchased products according to the global specification of Model 5.3. This `selection` event corresponds to one transition in the state-transition system of Figure 5.1.

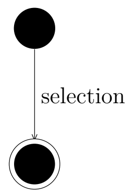


Figure 5.1 – The state-transition system for the `selection` event

Figure 5.1 defines one transition. The `selection` event, corresponding to the web service selecting the set of products, will be decomposed by refinement into other more concrete state-transition system. The resulting decompositions define different correct refinements corresponding to different compensation modes.

The figures presented here and in the next section use the statechart notation [Har87; OMG15]. Classical state-transition systems can be described and may be themselves decomposed into other state-transition systems that may be run in parallel (interleaved denoted by a dashed vertical line).

Equivalent compensation mode

The first compensation mode corresponds to *equivalence*. In this case, the logical expression is $\phi = \psi = true$. Figure 5.2 shows two possible refinements of the abstract *selection* service defined in Figure 5.1.

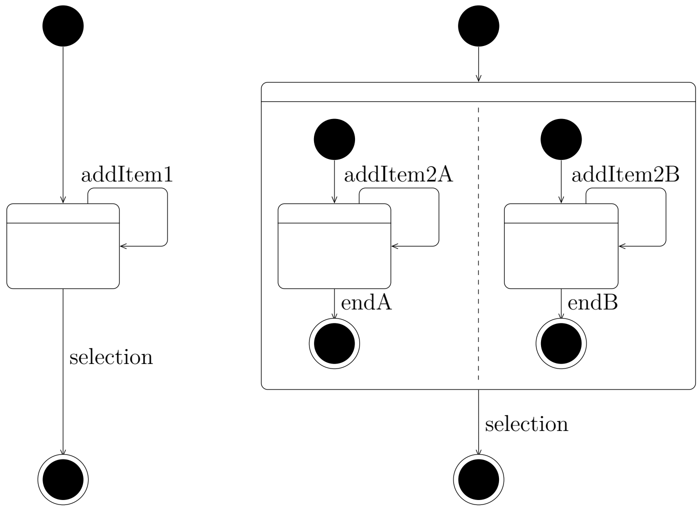
- The first one, on the left-hand side (see Figure 5.2a), corresponds to the case of a selection of a set of purchased products on a single website. The `addItem1` event loops until the products the end user whishes to purchase are selected.

- The second one, depicted on Figure 5.2b, corresponds to a selection of the purchased products realized on two different websites. Two interleaved processes (running in parallel; dashed lines) `addItem2A` and `addItem2B` are triggered. At the end, the set of selected purchased products is the union of the two sets obtained by each process.

In both cases, once the selection activity is completed, the `selection` event is completed.

Degraded compensation mode

The second compensation case deals with the degraded compensation mode. In our case study, we have described this situation by identifying lost products when the

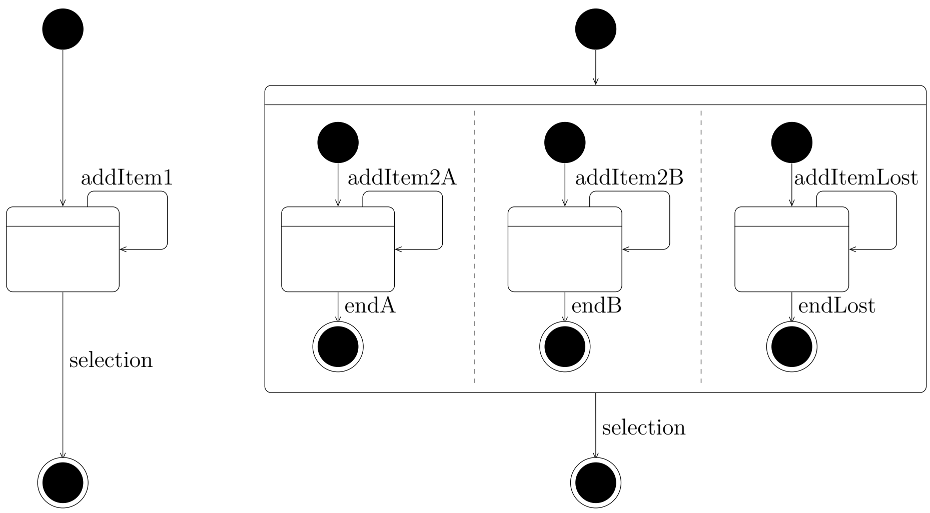


(a) Source service on a single website

(b) Target service on two websites

Figure 5.2 – Equivalent compensation mode

compensation holds. We assume that the selection of the purchased products on the two websites *WS1* and *WS2* does not contain all the specified set of products. The lost ones are collected by an abstract service using the `addItemLost` event.



(a) Source service for single website

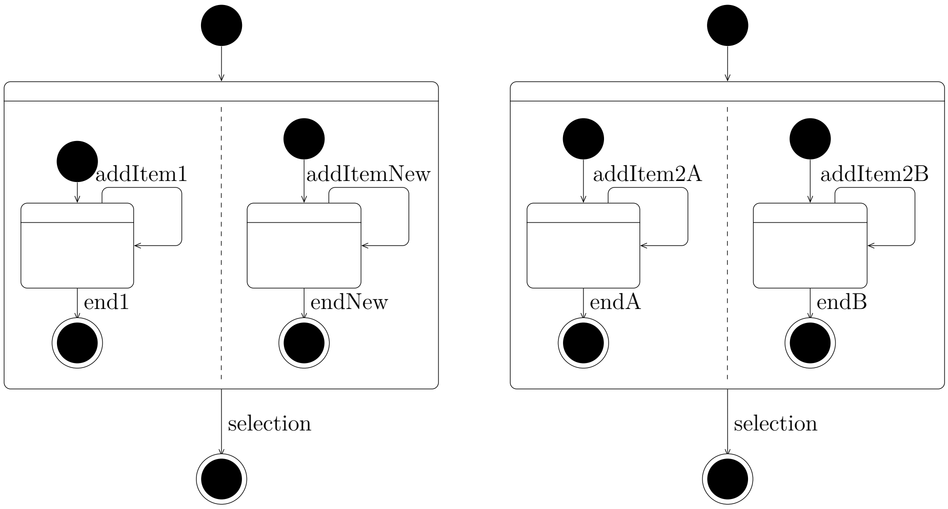
(b) Target service for two websites with an abstract service for lost products

Figure 5.3 – Degraded compensation mode

In both cases, once the *selection* activity is completed, the `selection` event is completed.

Upgraded compensation mode

The last compensation case concerns the definition of an upgraded mode. In our case study, this situation is shown on Figure 5.4. The source service, on a single website (Figure 5.4a) collects a set of products that is a subset of the set of specified products to be purchased. On the same figure, the other products are collected by an abstract service which adds new products using the event `addItemNew` loop. On the right side, the target service of Figure 5.4b collects the exact set of specified products.



(a) Source service for single website with an abstract service for new products

(b) Target service on two websites

Figure 5.4 – Upgraded compensation mode

In both cases, once the *selection* activity is completed, the `selection` event is completed.

5.2.5 Remark

In the upgraded and degraded modes, we have introduced what we call *abstract services* on one side or the other depending on the compensation case. These services are characterized by the ϕ and ψ properties of the horizontal invariant. These services are introduced to ensure a closed model where purchased products are modeled even if they are lost (in the case of the degraded mode) or new (in the case of upgraded mode). This is useful in the model to be able to precisely specify which products are lost or new. However, these abstract services would not appear in a concrete implementation of these models.

5.2.6 Cold start vs. warm start

Following the definition given in Section 2.1.1, *cold start* corresponds to the case where the restored state of the compensating service is the initial state. In other words, the compensating service runs from the beginning and erases the effects of the compensated service. This compensation mechanism is handled by the correct refinement. In case of compensation, any web service that refines the specification can be started or triggered from each initial state. The user will have to reenter all the input again.

Warm start corresponds to the case where the restored state of the compensating service is another state that collects the effects of the compensated service as much as possible. In other words, when a service is halted, the state variables of the compensating service are correctly updated by the values of the state variables of the compensated service. The *horizontal invariant* and the repairing event ensures that these values are safely copied. The way these variables are copied defines the *equivalent*, *degraded* or *upgraded* compensation modes. The use of modes to identify the running and compensating services guarantees the atomicity of the compensation although time is passing during compensation. The complexity of the compensation depends on the computations involved in the *horizontal invariant* expression defining the state restoration operation.

In the remainder, we deploy the defined methodology, in the case of discrete systems, for designing correct web services compensation. We show how the definition of a *horizontal invariant* makes it possible to define a compensation of a source service s by a target service t in the three cases shown in Section 5.2.3 and in the cold or warm start cases.

5.3 Deploying the stepwise methodology for defining consistent compensations with Event-B

The approach we define is a stepwise approach. This methodology allows a developer to design services compositions with correct compensations. By correctness, we mean not only the behavioral correctness, but also the functional correctness which is not addressed in most of the defined approaches of the literature. The proposed approach relies on refinement to characterize the correct compensating services on the one hand, and on invariant preservation to define relationships between a compensated service and a compensating service on the other hand.

The four steps of the defined methodology are described in the following.

5.3.1 Step 1. Composite web services as transition systems

First, a services composition is defined as a global specification. Then, a set of services compositions refining the defined global specification is given. Each services composition belonging to this set is seen as a transition system refining a global specification. At this step, we obtain a class of possible services compositions that simulate the global specification. When this process is repeated, a library of classes of services can be obtained. Each class characterizes all the services that refine the same activity.

5.3.2 Step 2. Introduction of failures and failure modes

Failures are introduced using explicit failure events. The effect of these events consists in suspending the current running service.

For this purpose, two modes are introduced using mode variables. A running mode stating that a service is currently active and a failure mode stating that a given service is in failure mode. The introduction of such modes contributes to the definition of the compensation order.

5.3.3 Step 3. Service recovery

Service recovery is performed thanks to a compensating event. This event selects the compensating service and transfers the control to this service.

This step requires the identification of the next state in the compensating service. Here, the defined gluing invariants are important, they define the next state in the compensating service. At this level, note that no selection criteria has been considered in this work, but this step can be completed by richer selection criteria, for example by exploiting quality of service properties. This aspect is out of scope of this work.

5.3.4 Step 4. Transferring control to the compensating service after failure

Finally, once the recovery state in the compensating service is known, it becomes possible to transfer the control to proceed with the execution of the composed service. This transfer is realized in two steps. First, the variables of the compensating service are updated and second, the compensating service mode is set to running mode. The next two sections show how this methodology is set up on the case study.

5.4 Case study: the root Event-B model

This section presents the formal Event-B root model associated to the case study defined in Section 3.1.4. This model represents the specification of a services composition. It will be refined later by several other refinement models that define possible implementations of this specification. Below, we give the context *C1* defining the relevant concepts needed to model the elements manipulated by the services and then the services composition is given by the *M0* abstract machine.

5.4.1 Context definition

The context *C1* of Model 5.1 defines the relevant sets for products (a finite set) and websites (at least two websites for the purpose of the case study). It also defines the *STOCKS* relation (Cartesian product). It relates websites to the products offered for purchasing by these websites. More precisely, it characterizes which products are available on a given website. Finally, P, denoting the set of products to be purchased, is defined.

```
Context C1
Sets
  PRODUCTS // all the products in the world
  SITES // all the sites in the world
Constants
  STOCKS
Axioms
  axm1: finite(PRODUCTS)
  axm2: finite(SITES)
  axm3: card(SITES) ≥ 2
  axm4: STOCKS = SITES × PRODUCTS
  axm5: P ⊆ PRODUCTS
End
```

Model 5.1 – The context *C1*

5.4.2 Model definition

The root model corresponding to the main Event-B machine *M0* in Models 5.2 and 5.3 formalizes the state-transition system of Figure 3.1. This machine is composed of the following elements.

- The *variables carts* denoting the cart containing the selected products and *seq* describing a sequencing variant on the events. These variables describe the state variables of the defined state-transition system. All these variables are defined in the `Variables` clause.

```
Machine M0 Sees C1
Variables P, cart, seq
Invariants
  inv1: carts ⊆ STOCKS
  inv2: seq < 4 ⇒ ran(cart) = P
  inv3: ∀p. p ∈ ran(cart) ⇒ card(cart⁻¹[{p}]) = 1
Variant seq
```

Model 5.2 – An Event-B model of the case study corresponding to Figure 3.1: variables and invariants

- The safety properties associated with the *selection* service are described by invariant properties in the `Invariants` clause. These properties, to be preserved by all the events, contain the typing properties for the state variables (*inv1*). Moreover, they state that:
  - *cart* contains the currently purchased products from websites;

- once the selection of products to be purchased is completed ($seq < 4$), the set of purchased products is the expected one (being P) by *inv2*;
- a product p in the set of purchased products *cart* is purchased only once, by *inv3*.

The property *inv2* of the `Invariants` clause guarantees that the set up specification of the web services composition correctly purchases the desired set of products P. $ran(cart) = P$ is true after triggering the `selection` event.

So, any refining behavior preserving such an invariant will be considered as a possible compensating services composition of the service composition defined by this specification. The definition of the invariant is fundamental in the correctness of the approach we propose.

```
Events
  Event Initialisation ≙
    Begin
      act1: cart := ∅
      act2: seq := 4
    End
  Event selection ≙
    Any someCart
    Where
      grd1: seq = 4
      grd2: someCart ⊆ SITES × P
      grd3: ran(someCart) = P
      grd4: ∀p. p ∈ ran(someCart) ⇒ card(someCart⁻¹[{p}]) = 1
    Then act1: seq := 3
         act2: cart := someCart
    End
  Event payment ≙
    Where grd1: seq = 3
    Then act1: seq := 2, ...
    End
  Event billing ≙
    Where grd1: seq = 2
    Then act1: seq := 1, ...
    End
  Event delivery ≙
    Where grd1: seq = 1
    Then act1: seq := 0, ...
    End
End
```

Model 5.3 – An Event-B model of the case study corresponding to Figure 3.1: the events encoding the activities (in machine *M0*)

- The `Initialisation` event is the first defined event (see Model 5.3). It sets up the *cart* to the empty set (meaning that no product is selected yet) and *seq* is assigned value 4, the number of sequential events. When $seq = 4$, it enforces `selection` to be the first event to be triggered. The scheduling of the events is guaranteed by *seq*;
- The other events define the sequence corresponding to the composed services. The description of these events and the triggering order defines a suitable composition. This description uses guards, a variant, non-determinism and interleaving semantics for events offered by Event-B to support either sequential or parallel composition.

  For the purpose of this case study, the following sequence of events has been defined (see Model 5.3) as follows.

  - The `selection` event sets up the *cart* (*act2*) to any cart *someCart* containing the specified set of products P whatever are the websites (*grd2* and *grd3*). It sets up the *seq* variable to 3 (*act1*) ensuring that the next triggered event will be the `payment` event.

    Let's observe that *grd3* and *grd4* guarantee that the invariant is preserved. Indeed, *grd3* guarantees that the set of purchased products is P, and *grd4* expresses that a product in *someCart* is purchased only once.

    Once the `selection` event is triggered, the set of purchased products corresponds to P.
  - When the product selection is completed, the `payment`, `invoicing` and `delivery` events, describing the corresponding activities, are ready to be triggered in this order thanks to the *seq* variant values occurring in the guards of these events.

  Note, that only the `selection` event is detailed. We do not give the details of the other events, since we illustrate service compensation on the *selection* service (activity).

5.4.3 Refining the root model

The root model represents the global specification of the defined services composition. Following the methodology described in Section 4.2, all the Event-B models that refine this root model are correct implementations of the defined specification. These implementations simulate, in the sense of the simulation relationship [Mil80; Mil89], the behavior of the specification.

Our approach exploits the refinement offered by Event-B. All the correct refinements of the global specification are candidates to implement the specification. This result gives us a way to characterize all the compensating services for a given specification. Indeed, it is enough to identify a refinement to get a possible compensating service. Refinement allows a developer to formally characterize a *class* of compensating services. But, yet, we did not describe the compensating process, we just identified the good compensating services.

In the following, we show how an implementation of the *selection* activity can be compensated by another implementation. We will exploit the refinement capability offered by Event-B.

5.5 A formal Event-B model for web services failure/compensation

This section applies the previous steps on the case study of Section 3.1.4. As mentioned previously, we are concerned with the compensation of the *selection* activity of the web services composition depicted on Figure 3.1 page 33 and whose Event-B model is given by the Models 5.2 and 5.3. Therefore, the other services *payment*, *invoicing* and *delivery* are not addressed in the developments presented below.

As mentioned in Section 5.2.4, two specific web services refining the *selection* activity are introduced.

- The first one, denoted *WS1*, allows a user to purchase the set of products P on a single website (namely $site_1$).
- The second one, denoted *WS2*, allows a user to purchase the set of products P using the combination of two different websites (namely $site_{2A}$ and $site_{2B}$).

Each of these services fills a cart of products denoted $cart_{WS1}$ for *WS1* and $cart_{WS2}$ for *WS2*.

The defined compensation considers that *WS1* is the running service. The `failure` and `compensate` events are introduced in order to switch from *WS1* to *WS2* in case of failure.

According to Section 5.2.4, this case study shows three compensation cases: equivalent, degraded and upgraded compensations. In the following, we describe in details how the defined methodology works for the case of *equivalent* compensation. The main development activities are described for the two other compensations cases (*degraded* and *upgraded* compensations).

5.5.1 Equivalent compensation: application to the case study

The *equivalent* compensation case corresponds to the refined `selection` event defined by the state-transition system depicted on Figure 5.2. The left and right sides of this figure describe the state-transition systems that behave equivalently from a functional point of view (the goal of the service).

Following the definition of the compensation given in Section 5.2.1, the horizontal invariant corresponding to the compensation depicted on Figure 5.2 is

$$cart_{WS1} = P \iff cart^{A}_{WS2} \cup cart^{B}_{WS2} = P$$

According to the identified compensation cases of Section 5.2.3, we can assert that the compensation is performed by an equivalent service (See Section 5.2.4 with $\phi = \psi = true$).

The equivalence relation in the previous expression enables us to repair both the *WS1* and *WS2* services. From left to right, *WS1* is compensated (repaired) by *WS2*. This expression splits $cart_{WS1}$ into two carts $cart^A_{WS2}$ and $cart^B_{WS2}$. From right to left, *WS2* is compensated (repaired) by *WS1* by the union of $cart^A_{WS2}$ and $cart^B_{WS2}$ carts in the cart $cart_{WS1}$.

The global system can continue the execution seamlessly without losing any product. Moreover, we guarantee the functional correctness of the global system through the proof of the refinement of the specification.

Having described the different resources needed to set up the compensation of *WS1* by *WS2*, we are ready to describe the whole Event-B development encoding this compensation following the stepwise methodology of Chapter 4 applied to this case.

Step 1. Composite web services as transition systems

A machine refining the *M0* machine is defined for each system. Two events (`selection_WS1` and `selection_WS2`) refining the `selection` event in the Event-B model of Model 5.3 are defined. They correspond to *WS1* and *WS2*. They are defined as follows.

1. The first refinement *R1*, described in Model 5.4, defines one possible web service implementing the *selection* activity in case of a single website $site_1$. It introduces a new event triggered as long as the cart $cart_{WS1}$ associated to the website $site_1$ does not contain the suited set P of products (*grd1* of the `addItem_WS1` event). The chosen product *item* is added to the cart (*act1*). Once the cart contains all the products of the set P, then, the event `selection_WS1` refining the `selection` event can be triggered, since its guard *grd1* becomes true.

   Note that this refinement introduces a new variable *sys*, acting as a mode variable, defining the current running system (here the web service *WS1* with one website).

2. The second refinement *R2*, described in Model 5.5, defines a second web service implementing the *selection* activity in the case of two websites $site_{2A}$ and $site_{2B}$. Here again, this refinement consists in introducing two events triggered as long as the union of the two carts $cart^A_{WS2}$ and $cart^B_{WS2}$ does not contain the set of all products P to be purchased (events `addItemA_WS2` and `addItemB_WS2`). In the same manner, once the cart contains all the products, the event `selection_WS2` refining the `selection` event can be triggered, since its guard *grd1* is true.

   Here again, note that this refinement introduces a new variable *sys*, acting as a mode variable, defining the current running system (here the web service *WS2* with two websites).

Event addItem_WS1 $\widehat{=}$
  Any $item$
  Where grd1: $item \in P \setminus \mathrm{ran}(cart_{WS1})$
    grd2: $sys = 1$
  Then act1: $cart_{WS1} := cart_{WS1} \cup \{site_1 \mapsto item\}$
  End
Event selection_WS1 **Refines** selection $\widehat{=}$
  Where grd1: $\mathrm{ran}(cart_{WS1}) = P$
    grd2: $sys = 1$
  Then act1: $cart := cart_{WS1}$
  End

Model 5.4 – Refinement of `selection` for a single website (machine *R1* refining *M0*)

Event addItemA_WS2 $\widehat{=}$
  Any $item$
  Where grd1: $item \in P \setminus \mathrm{ran}(cart^A_{WS2} \cup cart^B_{WS2})$
    grd2: $sys = 2$
  Then act1: $cart^A_{WS2} := cart^A_{WS2} \cup \{site_{2A} \mapsto item\}$
  End
Event addItemB_WS2 $\widehat{=}$
  Any $item$
  Where grd1: $item \in P \setminus \mathrm{ran}(cart^A_{WS2} \cup cart^B_{WS2})$
    grd2: $sys = 2$
  Then act1: $cart^B_{WS2} := cart^B_{WS2} \cup \{site_{2B} \mapsto item\}$
  End
Event selection_WS2 **Refines** selection $\widehat{=}$
  Where grd1: $\mathrm{ran}(cart^A_{WS2} \cup cart^B_{WS2}) = P$
    grd2: $sys = 2$
  Then act1: $cart := cart^A_{WS2} \cup cart^B_{WS2}$
  End

Model 5.5 – Refinement of `selection` for two websites(machine *R2* refining *M0*)

Step 2. Introduction of failures and failure modes

Failure modes are introduced by a new context *C11* extending the *C1* context (see Model 5.6). It defines the *FAILURE_MODES* set of modes and two constants indicating if a system is in a failure state or not. *axm1* states that they define a partition of the *FAILURE_MODES* set (*i.e.* *OK* and *NOK* are different).

Context C11 **Extends** C1
Sets $FAILURE\_MODES$
Constants OK, NOK
Axioms
  axm1: $\mathrm{partition}(FAILURE\_MODES, \{OK\}, \{NOK\})$
End

Model 5.6 – Introduction of a context for failure modes

A machine *R3* refining the *M0* machine is defined. A new variable *failureStatus* is introduced to complete the definition of modes. It records if the system is in a failure mode or not. *sys* still describes which web service is currently running among the available services. A new event, named `failure_WS1` is introduced. It is triggered when a failure occurs on *WS1*. Model 5.7 defines this event.

Event failure_WS1 $\widehat{=}$
  Where
    grd1: $sys = 1$
    grd2: $failureStatus = OK$
  Then
    act1: $failureStatus := NOK$
  End

Model 5.7 – Failure event (in machine *R3* refining *M0*)

The effect of this event is to switch the global web services composition from a normal mode to a failure mode (*act1*).

Step 3. Service recovery

At this level, the whole web services composition is halted (*grd2*). The repairing event exploiting the horizontal invariant can be triggered. Model 5.8 shows how the compensation is handled.

The `compensate_WS1_WS2` event copies the current state variables of the failed service (*act3* and *act4*) into the new state variables of the compensating service. The variable *sys* changes value to 2 (*WS2*) and the *failureStatus* is turned to an *OK* mode. At this stage, the compensating service is ready to run.

Event compensate_WS1_WS2 $\widehat{=}$
  Any $aCart^A_{WS2}$, $aCart^B_{WS2}$
  Where
    grd1: $sys = 1$
    grd2: $failureStatus = NOK$
    grd3 : $\mathbf{aCart^A_{WS2} \cup aCart^B_{WS2} = cart_{WS1}}$
    grd4: $aCart^A_{WS2} \cap aCart^B_{WS2} = \varnothing$
  Then
    act1: $sys := 2$
    act2: $failureStatus := OK$
    act3: $cart^A_{WS2} := aCart^A_{WS2}$
    act4: $cart^B_{WS2} := aCart^B_{WS2}$
  End

Model 5.8 – The compensating event exploiting the horizontal invariant (in machine *R3* refining *M0*)

Step 4. Transferring control to the compensating service after failure

At this level, compensation is completed. Indeed, the `compensate_WS1_WS2` event of Model 5.8 has set up to true all the conditions to trigger the `addItemA_WS2`,

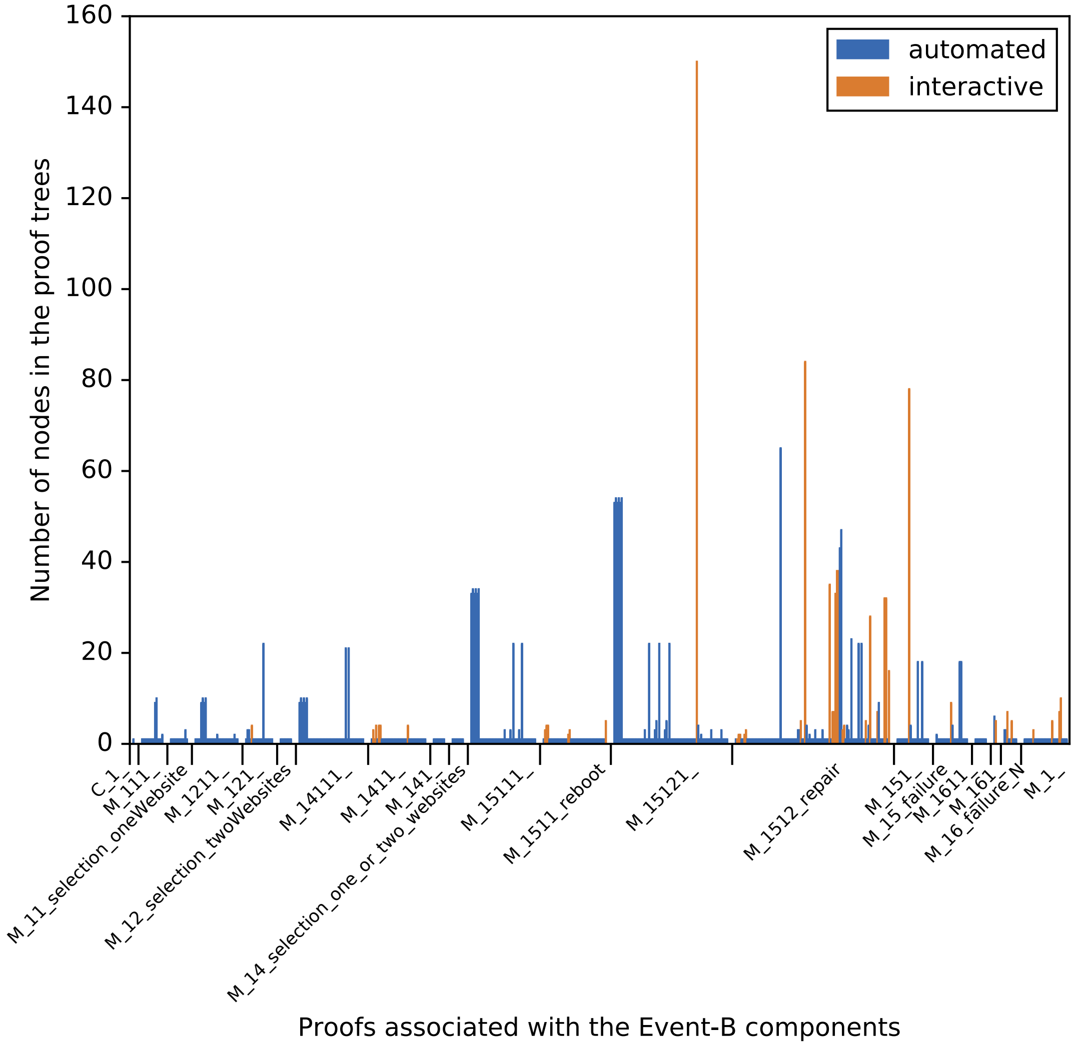


Figure 5.5 – Proofs size (number of nodes in the proof trees)

`addItemB_WS2` and `selection_WS2` events of the compensating service with two websites.

5.5.2 Some remarks

The previous development showed on a case study how the refinement offered by Event-B supports the definition of correct compensation mechanisms for web services compositions. It illustrated how the proposed methodology for system substitution that was defined in Chapter 4 applies to discrete system substitution. This development led to a completely proved formal development available in Appendix B.

Table 5.1 shows the results of the experiments we conducted within the Rodin Platform for Event-B. The presented development has been entirely encoded and proved. Deadlock freeness, correct behavior, refinements and compensation correctness properties have all been proved. The results show that few proof

Table 5.1 – Statistics related to the proofs performed with the Rodin Platform

Machine	Automated PO	Interactive PO
Root Machine *M0*	27	4
Refinement for *WS1* (*R1*)	27	1
Refinement for *WS2* (*R2*)	56	1
Refinement with compensation (*R3*)	214	16
Total	324	22

obligations (PO) required interactive proofs (22 among 324 generated POs).

The sizes of the various proofs for the various machines and contexts are available in Figure 5.5.

5.6 Other compensation cases: upgraded and degraded

As mentioned previously, in this section, we consider the two remaining compensation cases among the three identified ones (*i.e. upgraded* and *degraded* modes).

5.6.1 Compensation in presence of degrading services

The second case of compensation is the degraded case. It corresponds to the case where products are lost when a compensation is performed. This case is depicted in Section 5.2.4 on Figure 5.3. The following horizontal invariant is introduced to characterize this situation.

$$cart_{WS1} = P \iff cart^{A}_{WS2} \cup cart^{B}_{WS2} \cup Lost = P$$

It states that the compensating service looses a set of products *Lost* that were originally in the compensated service's cart. We have followed the same methodology as for the equivalent case. The main difference occurs in *Step 3*, where the repairing (compensating) event must guarantee the horizontal invariant.

The repairing event exploiting the horizontal invariant can be triggered. Model 5.9 shows how the compensation is handled.

The `compensate_WS1_WS2_deg` event splits the current cart of the failed service (*act3* and *act4*) into the new state variables of the compensating service. A new state variable, the set *Lost*, is defined in the compensating service. This variable is introduced to guarantee that the horizontal invariant holds. The other substitutions behave as for the equivalent compensation case.

```
Event compensate_WS1_WS2_deg ≙
  Any aCart^A_WS2
      aCart^B_WS2
      aLost // products that will be lost
  Where
    grd1: sys = 1
    grd2: failureStatus = NOK
    grd3: aCart^A_WS2 ∪ aCart^B_WS2 ∪ aLost = cart_WS1
    grd4: aCart^A_WS2 ∩ aCart^B_WS2 = ∅
    grd5: aCart^A_WS2 ∩ aLost = ∅
    grd6: aCart^B_WS2 ∩ aLost = ∅
  Then
    act1: sys := 2
    act2: failureStatus := OK
    act3: cart^A_WS2 := aCart^A_WS2
    act4: cart^B_WS2 := aCart^B_WS2
    act5: Lost := aLost
  End
```

Model 5.9 – Compensating event and horizontal invariant (degraded case)

5.6.2 Compensation in presence of upgrading services

The third case of compensation is the upgraded case which corresponds to the case where more products than specified are purchased. This case is depicted in Section 5.2.4 on Figure 5.4. The following horizontal invariant is introduced to characterize this situation.

$$cart_{WS1} \cup New = P \Longleftrightarrow cart^A_{WS2} \cup cart^B_{WS2} = P$$

It states that the compensating service offers a *New* set of products that were not originally in the compensated service's cart. Again, we have followed the same methodology as for the equivalent case. The main difference occurs in *Step 3* where the repairing (compensating) event must guarantee the horizontal invariant. This repairing event exploiting the horizontal invariant can be triggered. Model 5.10 shows how the compensation is handled.

The `compensate_WS1_WS2_upg` event copies the current state variables of the failed service (*act3* and *act4*) into the new state variables of the compensating service, but the *New* set is included in $cart^A_{WS2} \cup cart^B_{WS2}$. The other substitutions behave as for the equivalent case.

If applied to two web services, it corresponds to the case where the compensating service offers more functionalities than offered by the compensated service. For example, when purchasing flight tickets, one can use a website that offers more products to purchase like booking hotel rooms, car rentals, *etc.*

Event compensate_WS1_WS2_upg $\widehat{=}$
Any $aCart^{A}_{WS2}$
$aCart^{B}_{WS2}$
aNew // *products that will be added*
Where
grd1: $sys = 1$
grd2: $failureStatus = NOK$
grd3: $\mathbf{aCart^{A}_{WS2} \cup aCart^{B}_{WS2} = cart_{WS1} \cup aNew}$
grd4: $aCart^{A}_{WS2} \cap aCart^{B}_{WS2} = \varnothing$
Then
act1: $sys := 2$
act2: $failureStatus := OK$
act3: $cart^{A}_{WS2} := aCart^{A}_{WS2}$
act4: $cart^{B}_{WS2} := aCart^{B}_{WS2}$
End

Model 5.10 – Compensating event and horizontal invariant (upgraded case)

5.7 Conclusion

Several approaches have been defined and succeeded in verifying correct behaviors of composite web services compensations.

Due to the abstraction of services input/output to avoid state number explosion, little attention has been paid to the verification of functional correctness of service compensation.

In this chapter, we have applied our methodology for correct substitution to the discrete case of service compensation. It has been published in [BAP15] and [BAP17].

The approach we have developed in this chapter relies on the definition of *horizontal invariants* that establish a relation between services' states. This relation leads to the definition of a class of equivalent services with respect to the defined relation (loose coupling of services). Each service refining (implementing) a given activity is a candidate to compensate a service. Indeed, each service refining a service specification is a candidate for correct compensation in a cold start context.

Then, a stepwise methodology consisting in gradually introducing failure and compensating events has been defined. It is compatible with the definition of compensation available in languages like BPEL. We have shown on a case study how this approach works and a whole Event-B development has been described. Moreover, the proposed approach also addressed two major aspects of compensation.

- The first one is the capability to make compensation at runtime. Indeed, the definition of horizontal invariants makes it possible to define compensation events that repair the suspended activity and switch from a failed service to a compensating one by affecting its variables consistently.

- The second key point concerns the nature of the horizontal invariant. Indeed, equivalent, degraded or upgraded compensations can be expressed. The equivalence relation defined allows a developer to check the quality of the compensating service. This situation has been shown on three compensation cases whose definition relies on the provided horizontal invariant.

Then, the defined compensation mechanism supports a dynamic compensation. When the horizontal invariant is correctly chosen (by correct, we mean that it preserves the one of the original specification), then the repairing event recovers the state of the compensated service in the compensating service. This feature is relevant for defining compensation on-the-fly during service orchestration.

Finally, the proposed approach promotes openness. Indeed, the definition of compensating services can be done dynamically. It requires adding new compensating services to the class of services, provided they define a correct refinement of the compensated activity. In this case, the service may be chosen to compensate a failed service. In other words, refinement allows a service designer to characterize a whole set (a class) of compensating services.

In this chapter we detailed a compensation mechanism based on discrete substitution using our modeling framework. In the next chapter (Chapter 6), we will introduce the modeling of continuous systems in Event-B. Then in the following chapter (Chapter 7), we will present our work on continuous system substitution.

6 Hybrid systems: Continuous to discrete models

Chapter organization. Section 6.2 overviews the addressed problem of discretization. The refinement strategy for any continuous function together with the corresponding requirements are given in Section 6.3, while the complete Event-B development handling these requirements is provided in Section 6.4.

6.1 Introduction

Before addressing the case of non-instantaneous (non-atomic) system substitution, we first study how systems with models relying on continuous time over real numbers can be modeled using the refinement and proof method Event-B. These models allow designers to describe hybrid systems. We show how, under some hypotheses, continuous systems descriptions are correctly discretized.

In the past years, several approaches relying on formal methods, like Hybrid automata [Hen00] and model checking [Alu11], have been set up to describe the behavior of the software controllers. Our proposal focuses on the synthesis of correct discrete controllers for hybrid systems.

Objective of this chapter. This chapter shows how proof and refinement based approaches handle the development of a correct-by-construction discrete controller starting from a continuous time function specification of the continuous controller. A complete incremental development relying on a theory of reals is conducted to synthesize a correct discretization of a continuous function. The approach exploits an axiomatization of mathematical reals. It maintains a safety invariant characterizing the physical plant of the studied system. Such an invariant defines a safety envelope (which we also named *safety corridor*) modeling a *stability property* in which the system must evolve *i.e.* for a continuous function f, we write $\forall t \in \mathbb{R}^+, f(t) \in [m, M]$ where t is a continuous time parameter belonging to $\mathbb{R}^+$ and the reals m and M define respectively minimum and maximum values in $\mathbb{R}^+$ ensuring a correct behavior of the physical plant, whose behavior is modeled by the function f. In general, these values are the result of the physics of the studied system. The Event-B method is used to handle such formal developments. We illustrate our proposal with the development of a simple stability controller for a generic plant model. Next, we will address system substitution where systems are characterized by such models.

6.2 Discretization of continuous functions

The behavior of many systems can be characterized by three phases: the initial boot, the nominal behavior, and the halting of the system. Several CPS integrating physical plants and software controllers follow this state evolution pattern. Examples of such systems are energy production systems, smart systems, medical systems, *etc*. These systems are usually modeled by differential equations specifying continuous time functions. In order to design a software controller running on discrete time steps to handle their behavior, one has to discretize these continuous functions. The main safety property concerns stability where the function values shall be maintained inside a safety envelope, *i.e.* an interval of correct values, called *corridor*.

The correct implementation of such continuous functions is a key point in ensuring CPS safety. They shall be correctly discretized *i.e.*, guarantee that the discrete behavior simulates the continuous one. In other words, the continuous states existing between two observed consecutive states of the discretization shall also be in the safety corridor.

To achieve this goal, we follow a correct-by-construction approach based on a formal development of *any* continuous function discretization, making our development reusable and scalable. The approach relies on refinement and on the preservation of invariants. Discretization information is incrementally added while moving from the continuous level to the discrete one. Event-B [Abr10] and the Rodin Platform [Abr+10] have been set up to handle the developments.

6.3 Refinement strategy

We sketch here the mathematical model and the specification of the system behavior. Following the approach defined in [SAZ14], the adopted refinement strategy consists in three steps: first, as shown in Figure 6.1, we use three states to define a simple abstract controller that models the system by introducing modes; then, in a first refinement, we introduce a continuous controller characterizing its behaviors with a continuous function; finally, a second refinement builds a discrete controller of the system.

6.3.1 The illustrating system

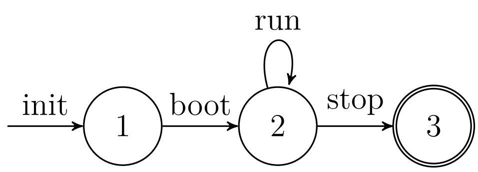


Figure 6.1 – Controller automaton

The behavior of the considered system is defined through three phases. Figure 6.1 depicts its general behavior using a state-transition system. First, it is booted (transition *boot* from state 1 to state 2). After a while (time passing), once in state 2, it becomes operational in a nominal mode (*run* transition). Then, it stays a given amount of time in the nominal or running mode. When in nominal mode, it may be halted (*stop* transition from state 2 to state 3) for example in case a failure occurs or for maintenance purposes. This behavior is the one of a simple *abstract* system controller. We have considered that, when booting, the system cannot be stopped until it reaches the nominal mode. Other complex scenarios can be defined with more complex transition systems.

Table 6.1 – Requirements for the top level

At any time, in any mode, the output value of the controlled system shall be less or equal to M.	Req.1
At any time, in running mode, the output value of the controlled system shall belong to an interval $[m, M]$.	Req.2
At any time, in running mode, if any future output value of the controlled system does not belong to an interval $[m, M]$, then the system is stopped.	Req.3

In order to guarantee a correct behavior of the system, the previously defined controller shall fulfill the requirements from Table 6.1. These ones ensure that the system is correctly controlled. For example, an energy production system requires that the power produced by a given system belongs to a specific interval or a pacemaker must be pacing when a sensed signal belongs to another specific interval.

6.3.2 Continuous controller

After modeling the system at an abstract level using three states, the continuous controller is introduced through the definition of a continuous function of the continuous time $f : \mathbb{R}^+ \to \mathbb{R}$ to characterize the behavior of the system.

The requirements identified in the previous section, are rewritten (refined) to handle the introduced continuous function behavior (see Table 6.2).

Table 6.2 – Requirements for the first refinement

$m < M$	Req.0
$\forall t \in \mathbb{R}^+, f(t) \leq M$	Req.1
$\forall t \in \mathbb{R}^+, state(t) = 2 \Rightarrow f(t) \in [m, M]$	Req.2.1
$\forall t_1, t_2 \in \mathbb{R}^+, t_1 < t_2, state(t_1) = 2 \wedge f(t_2) \in [m, M] \Rightarrow state(t_2) \in \{2, 3\}$	Req.2.2
$\forall t_1, t_2 \in \mathbb{R}^+, t_1 < t_2, state(t_1) = 2 \wedge f(t_2) \notin [m, M] \Rightarrow state(t_2) = 3$	Req.3

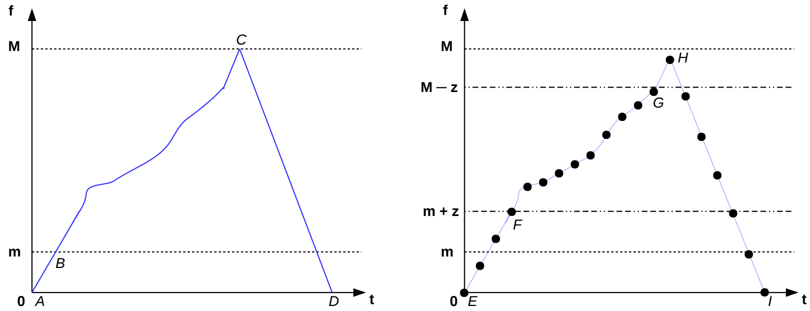
The control action over this system is a simple one. It consists in shutting down the system if the value of f goes out of range. The obtained continuous controller corresponds to a refinement of the abstract one from the previous section, it is described by a hybrid automaton. We are aware that the control actions of the defined system are very simple. Our objective is to show how a controller (characterized by a simple state transition system) and a physical plant (characterized by a continuous function) can be formally integrated into a single Event-B formal development encoding incrementally a hybrid automaton.

One possible behavior corresponding to the previous description is depicted by the graph in Figure 6.2a. The system is initialized (at point A corresponding to the transition `init` to enter state 1). It reaches the running mode state at point B (corresponding to the event `boot` and entering state 2). The system remains in the safety corridor (between m and M in state 2). When point C is reached, the controller switches its state from state 2 to state 3 with the transition `stop` in order to prevent f from going over the threshold M. The system is then halted to reach point D (corresponding to state 3).

6.3.3 Discrete controller

In order to implement the previous controller, we need to discretize the observation of the system behavior. In practice, when using computers to implement such controllers, time is observed according to specific clocks and periods or frequencies. In other words, observations are discrete and depend on the available clocks. Therefore, it is mandatory to define a correct time discretization that preserves the continuous behavior introduced previously. This preservation entails the introduction of other requirements (hypotheses) on the defined continuous function. Note that, in practice, these requirements correspond to requirements issued from the physical plant.

We introduce a margin allowing the controller to anticipate the next observable behavior before an incorrect behavior occurs. Let z be this margin. z is defined such that the evolution of the function f between two observed consecutive instants t_i and t_{i+1} shall not be greater than z.



(a) Continuous controller (b) Discrete controller

Figure 6.2 – Examples of the evolution of the function f

In order to formally define z, we first declare δt the fixed discretization interval, *i.e.* $\delta t > 0$ and $\forall i \in \mathbb{N}, \delta t = t_{i+1} - t_i$ and $\forall i \in \mathbb{N}, t_i = i \times \delta t$. Because of the physical nature of the system, we assume the function f to be Lipschitz continuous (the differential of f is bounded by a constant K, called the Lipschitz constant):

$$\exists K \in \mathbb{R}^+, \quad \forall t_1, t_2 \in \mathbb{R}^+, \quad |f(t_1) - f(t_2)| \leq K \times |t_1 - t_2|$$

We can assume that there exists z such that:

$$\forall t \in \mathbb{R}^+, \quad |f(t) - f(t + \delta t)| \leq z$$

It is possible to derive the property related to the bounded variation of the function f inside a discrete interval as follows:

$$\forall i \in \mathbb{N}, \quad \forall t \in [t_i, t_{i+1}], \quad |f(t_i) - f(t)| \leq z$$

Finally, we obtain a *safe progress* property stating that if the value of f is in the $[m + z, M - z]$ interval, then, the safety property $f(t) \in [m, M]$ is preserved until the next discrete instant:

$$\boxed{\forall i \in \mathbb{N}, \quad f(t_i) \in [m + z, M - z] \Rightarrow \forall t \in [t_i, t_{i+1}],\ f(t) \in [m, M]}$$

Additionally, for the problem to be well-defined, we impose that δt be small enough so that the property $m + z < M - z$ holds.

The set $\mathbb{D}$ of observation instants can be defined as:

$$\mathbb{D} = \{t_i \mid t_i \in \mathbb{R} \wedge i \in \mathbb{N} \wedge t_i = i \times \delta t\}$$

As a consequence of this definition, the safety corridor becomes the interval $[m + z, M - z]$. Moreover, it becomes possible to observe, in the *running* mode, two consecutive instants t_i and t_{i+1} such that:

$$\begin{cases} f(t_i) \in [m + z, M - z] \\ f(t_{i+1}) \notin [m + z, M - z] \\ f(t_{i+1}) \in [m, M] \end{cases}$$

Table 6.3 – Requirements for the second refinement

	$z > 0 \wedge m + z < M - z$	Req.0
	$\forall t_i \in \mathbb{D}, f(t_i) \leq M$	Req.1
	$\forall t_i \in \mathbb{D}, state(t_i) = 2 \Rightarrow f(t_i) \in [m + z, M - z]$	Req.2.1
	$\forall t_i \in \mathbb{D}, state(t_i) = 2 \wedge f(t_i + \delta t) \in [m, M] \Rightarrow state(t_i + \delta t) \in \{2, 3\}$	
$\Leftrightarrow$	$\forall t_i \in \mathbb{D}, state(t_i) = 2 \wedge f(t_{i+1}) \in [m, M] \Rightarrow state(t_{i+1}) \in \{2, 3\}$	Req.2.2
$\Leftrightarrow$	$\forall n \in \mathbb{N}, state(n\,\delta t) = 2 \wedge f((n + 1)\,\delta t) \in [m, M] \Rightarrow state((n + 1)\,\delta t) \in \{2, 3\}$	
	$\forall t_i \in \mathbb{D}, state(t_i) = 2 \wedge f(t_i + \delta t) \notin [m + z, M - z] \Rightarrow state(t_i + \delta t) = 3$	
$\Leftrightarrow$	$\forall t_i \in \mathbb{D}, state(t_i) = 2 \wedge f(t_{i+1}) \notin [m + z, M - z] \Rightarrow state(t_{i+1}) = 3$	Req.3
$\Leftrightarrow$	$\forall n \in \mathbb{N}, state(n\,\delta t) = 2 \wedge f((n + 1)\,\delta t) \notin [m + z, M - z]$ $\Rightarrow state((n + 1)\,\delta t) = 3$	

This condition characterizes a behavior that exits the safety corridor and thus it identifies the condition for stopping the system (*i.e.* moving to a stopping mode). Again, the previous requirements are refined to consider the discretization of time, using the two new parameters z and δt, and $\mathbb{D}$ (Table 6.3).

The safety margin z is defined such that if $f(n \times \delta t)$ is in $[m + z, M - z]$ then the value of f observed by the controller, $f((n + 1) \times \delta t)$, is in $[m, M]$. The defined discretization guarantees that *Req.2.1* is fulfilled until the next discrete instant due to $\forall n \in \mathbb{N}, \quad \forall t \in [n \times \delta t, (n + 1) \times \delta t], \quad |f(t) - f(n\,\delta t)| \leq z$. If the controller observes a value in $[m, m + z[$ or in $]M - z, M]$, it shuts the system down because, the value might be out of range (*Req.3*) in the next step.

6.3.4 Top-down development

According to the previous definitions, refinement starts from a generic definition of the system with the three identified events. The first refinement introduces the continuous function and the corresponding requirements of Table 6.2. We start with a continuous model M_c of the system, describing the complete relevant physical behavior of the system. Then a second refinement defines the discrete model M_d of the behavior correctly glued with the continuous one. Here, the refined requirements of Table 6.3 are taken into account. Gluing invariants, formalizing the refined requirements, are introduced in order to preserve the proofs and the behavior of the abstraction. When proving the refinement, we formally establish that our discrete model is a correct implementation of the desired continuous behavior (the specification).

To summarize, in M_c, the continuous function $f_c : \mathbb{R} \longrightarrow \mathbb{R}$ is considered. In M_d, we introduce a discrete function $f_d : \mathbb{N} \longrightarrow \mathbb{R}$, where $i \in \mathbb{N}$ is an instant and δt is the time discretization interval duration. The functions f_d and f_c are glued by the following property: $\forall n \in 0..i,\ f_c(n \times \delta t) = f_d(n)$.

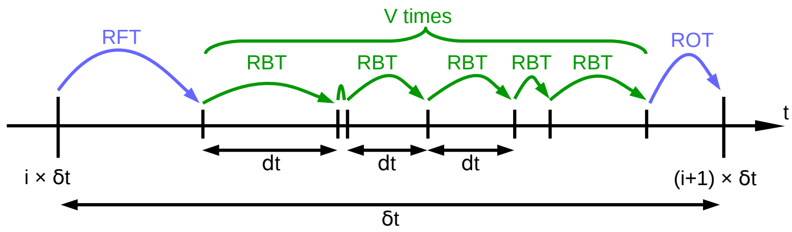


Figure 6.3 – Collapsing continuous time micro steps into a discrete time macro step

6.3.5 About the modeling of time

In order to reduce the complexity of the proof of the discretization refinement corresponding to the introduction of f_d, we have split the behavior of f_c during an i^{th} discrete *macro step* $[t_i,\ (t_i + \delta t)]$ into three kinds of smaller finite discrete *micro steps* (see Figure 6.3). For example, at the running state (or nominal phase), we define the following micro steps.

1. `RFT`: *run from tick* is the first micro step inside a macro step starting at a *tick* (a discrete time $t_i = i \times \delta t$). Its duration is strictly smaller than δt.

2. `RBT`: *run between ticks* is a micro step strictly in the macro step (not the first nor the last micro step in a macro step). Its duration is denoted $dt > 0$. A macro step contains V occurrences of such micro steps.

3. `ROT`: *run on ticks* is the last micro step in the macro step.

Because δt the duration of the steps can be infinitely small, there could be an infinite number of steps: this is called the Zeno problem. It is avoided here by guaranteeing that the number of micro steps of type `RBT` is finite, and that $dt > 0$. From a modeling point of view, it will be formalized as a decreasing variant (natural number V in $\mathbb{N}$). The trace of micro steps between t_i and $t_{i+1} = t_i + \delta t$ is defined as $\texttt{RFT}\ (\texttt{RBT})^V\ \texttt{ROT}$. The correctness of the discretization ensures that we can take a finite number that depends on the physical parameters of the system.

Our Event-B models introduce events aligned with these macro and micro steps either in the continuous case of in the discrete one.

6.4 A formal development of a discrete controller with Event-B

Our developments expressed using Event-B follow the refinement strategy defined in Section 6.3. Following [SAZ14], three development steps have been used. Contexts and machines are defined according to Figure 6.4.

6.4.1 Abstract machine: the top-level specification

The top-level specification introduces the abstract controller with three events according to Figure 6.1.

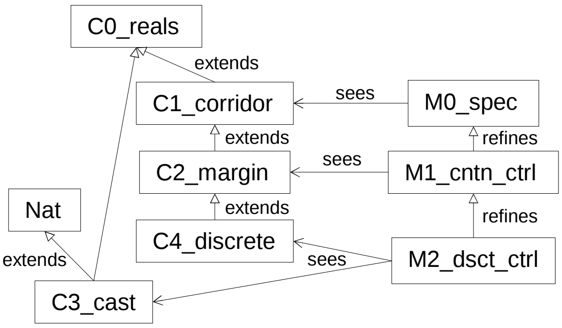


Figure 6.4 – Project structure

Needed theories

To be able to handle mathematical real numbers and the corresponding theory, we have defined the context *C0_reals* which uses the theory defining mathematical reals. Model 6.1 gives an extract of this context with axioms and theorems.

Several other axioms and theorems have been defined and proven. We show an extract of this theory (see the Appendix C). As mentioned in Section 1.8, specific operators for manipulating reals are used.

A second context defines the safety corridor with the values of m and M. Model 6.2 defines this context *C1_corridor* extending the context *C0_reals*.

```
Context C0_reals
Constants REAL_POS, REAL_STR_POS
Axioms
  def01: REAL_POS={x | x ∈ REAL ∧leq(zero,x)} // ''leq'' is ≤ for reals
  def02: REAL_STR_POS={x | x ∈ REAL ∧smr(zero,x)} // ''smr'' is < for reals
  ...
Theorems
  thm01: ∀a,b · ( a ∈ REAL ∧b ∈ REAL ) ⇒( smr(zero,b) ⇒smr(a sub b , a) )
  thm02: ∀a,b · smr(a,b) ⇔¬ leq(b,a)
  ...
End
```

Model 6.1 – Part of context *C0_reals*

```
Context C1_corridor
Extends C0_reals
Constants m, M
Axioms
  axm01: m ∈ REAL_STR_POS
  axm02: M ∈ REAL_STR_POS
  axm03: smr(m,M)
End
```

Model 6.2 – Part of context *C1_corridor*

The top-level Event-B machine

It defines the global continuous variables issued from the controlled system. The machine introduces invariant *inv03*, guaranteeing *Req.1* and *Req.2.1* stating that in *running* mode (identified by $active = \text{TRUE}$), the real values of the continuous variables (defining the values of a continuous function introduced in the first refinement) *fv* shall be correct. This machine also models the abstract controller with three events `boot`, `run` and `stop` corresponding to the transition system of Figure 6.1. These events manipulate *fv* the real positive value of the continuous variables corresponding to the current continuous values without explicit definition of a function f.

Model 6.3 gives an extract of the top specification machine *M0_spec*.

```
Machine M0_spec Sees C1_corridor
Variables fv, active
Invariants
  inv01: fv ∈ REAL_POS
  inv02: active ∈ BOOL
  inv03: active = TRUE ⇒leq(m,fv) ∧leq(fv,M)
  inv04: active = FALSE ⇒fv = zero
Events
  Event Initialisation ≙
    Begin
      act01: active := FALSE
      act02: fv := zero
    End
  Event boot ≙ ...
  Event run ≙
    Any new_fv Where
      grd01: active = TRUE
      grd02: new_fv ∈ REAL_POS
      grd03: leq(m,new_fv) ∧leq(new_fv,M) // new_fv ∈ [m,M]
    Then
      act01: fv := new_fv
    End
  Event stop ≙ ...
End
```

Model 6.3 – Extract of machine *M0_spec*

Only details for the event `run` are given here. The complete Event-B developments can be found in Appendix C. Therefore, *Req.3* is not explicitly handled in this description, it mainly concerns the `stop` event.

6.4.2 The first refinement: introducing continuous functions

Needed theories

As shown on Figure 6.4, the context *C2_margin* introducing margin z is defined. Note that *axm02* corresponds to the requirement *Req.0*.

```
Context C2_margin Extends C1_corridor
Constants z
Axioms
  axm01: z ∈ REAL_POS // z ∈ R+
  axm02: gtr(M sub m , (one plus one) mult z) // M−m > 2∗z
End
```

Model 6.4 – Extract of context *C2_margin*

The Event-B first refinement with continuous functions

The first refinement *M1_cntn_ctrl* of the controller explicitly introduces:

- the continuous function *fc* producing the values *fv* of the abstract machine and the corresponding invariant *prop01*,
- continuous time with the current instant noted *now*,
- an important invariant *glue01* gluing the continuous variables of the abstraction with the continuous function defined on continuous time $fv = fc(now)$,
- the variable *active_t* recording the continuous time where the system enters a running mode and the corresponding invariants *glue02*, *glue03* and *glue04* gluing the behavior of *active_t* with the *active* boolean variable of the top level specification.

The events of the *M1_cntn_ctrl* machine refine the ones of the top level specification. The `boot` event fixes the value of *active_t* and the `run` event builds the continuous function *fc* with steps of duration *dt*. *fc* becomes the function *nfc*, acting until $now + dt$ instant.

The current instant *now* is increased by the step duration *dt* as well. The guards of the event `run` introduce the relevant conditions to trigger this event.

Note that during the time interval *dt*, the function *fc* shall be continuous and monotonic so that its value is never outside the safety corridor (*grd09* to *grd11*). This condition is fundamental when the function is discretized. Thus, *grd09* through *grd12* guarantee the requirement *Req.2.2* and are of particular importance when discretizing.

```
Machine M1_cntn_ctrl Refines M0_spec Sees C2_margin
Variables fv, active, fc, now, active_t
Invariants
  type01: now ∈ REAL_POS
  type02: fc ∈ REAL_POS →REAL_POS
  type03: active_t ∈ REAL_POS
  prop01: cnt_int(fc, zero, now) // fc is continuous on [0,now]
  glue01: fv = fc(now)
  glue02: active = TRUE ⇒( ∀t ·t ∈ REAL ∧leq(active_t,t) ∧leq(t,now) ⇒
                        ( leq(m plus z , fc(t)) ∧ leq(fc(t) , M sub z) ))
  glue03: ∀t · t ∈ REAL ∧leq(zero,t) ∧ leq(t,now) ⇒leq(fc(t),M)
  glue04: active = TRUE ⇒leq(active_t,now)
Events
  Event Initialisation ≙ ...
  Event boot ≙ Refines boot ...
    Then
      ...
      act04: now := now plus dt
      act05: active_t := now plus dt
    End
  Event run ≙ Refines run
    Any
      dt, nfc, new_fv
    Where
      ...
      grd04: dt ∈ REAL_STR_POS // dt > 0
      grd05: nfc ∈ REAL_POS ⇸REAL_POS
      grd06: dom(nfc) = {t | t ∈ REAL ∧leq(now,t) ∧leq(t , now plus dt)}
                                            // dom(nf) = [now,now+dt]
      grd07: nfc(now) = fc(now)
      grd08: nfc(now plus dt) = new_fv
      grd09: leq(fv,new_fv) ⇒(∀ t1,t2 · t1 ∈ dom(nfc) ∧t2 ∈ dom(nfc)
                                   ∧ leq(t1,t2) ⇒leq(nfc(t1) , nfc(t2)))
                  // nfc is monotonic on [t1,t2]
      grd11: leq(new_fv,fv) ⇒(∀ t1,t2 · t1 ∈ dom(nfc) ∧t2 ∈ dom(nfc)
                                   ∧ leq(t1,t2) ⇒leq(nfc(t2) , nfc(t1)))
      grd10: cnt_int(nfc , now , now plus dt) // continuous on [now,now+dt]
      grd12: ∀t · t ∈ dom(nfc) ⇒leq(m plus z , nfc(t) ∧ leq(nfc(t) , M sub z)
    Then
      ...
      act02: now := now plus dt
      act03: fc := fc ⊲nfc
    End
  Event stop ≙ Refines stop...
End
```

Model 6.5 – Extract of machine *M1_cntn_ctrl*

6.4.3 The second refinement: introducing discrete representation

This refinement introduces the discretization function *fd* corresponding to the continuous function *fc* on each discrete observed instants. This fundamental property corresponds to requirement *Req.2.2* of Table 6.3. It is expressed by the invariants gluing the continuous controller and the discrete controller. It links the continuous f_c and discrete f_d functions by the property $\forall n \in 0\,..\,i \Rightarrow f_c(n \times \delta t) = f_d(n)$ and is represented in invariant *glue01*.

Needed theories

Two contexts are introduced. The first context *C3_cast* is a technical context related to casting reals and integers (see Section 1.8.3). For example, the invariant $\forall n \in 0\,..\,i \Rightarrow f_c(n \times \delta t) = f_d(n)$ corresponding to *glue01* is written as:
`∀ n · n ∈ 0..i ⇒ fc(cast(n) mult tstep) = fd(n).`

```
Context C3_cast Extends C0_reals, Nat
Constants cast
Axioms
  axm01: cast ∈ ℕ→REAL_POS     // type
  axm02: cast(0) = zero    // initial case
  axm03: ∀a · a ∈ ℕ ⇒         // induction case
             (cast(a+1) = cast(a) plus one)
Theorems
  ...
  thm11: ∀a,b · (a ∈ ℕ ∧ b ∈ ℕ) // equiv. over '<'
             ⇒(a < b ⇔smr(cast(a),cast(b)))
  thm12: ∀a,b · (a ∈ ℕ ∧ b ∈ ℕ) // equiv. over '='
             ⇒(a = b ⇔cast(a) = cast(b))
  thm13: cast ∈ ℕ ⤖ cast[ℕ] // cast is a bijection
  ...
End
```

Model 6.6 – Definition and properties of the *cast* function (reminder)

The last context *C4_discrete* introduces the discrete time macro steps duration *tstep* corresponding to δt on Figure 6.3 and the values `RBT` and `RV` (`run_variant`) to identify the different events corresponding to the `run` event. It also defines the *max_df* constant corresponding to the maximum evolution of the function in a macro step, which is never more than margin z (*axm03*). This assumption usually comes from the conditions on the physical plant.

```
Context C4_discrete Extends C2_margin
Sets VT
Constants
  tstep     //  discrete  time step  duration  (δt)
  max_df   //  maximum delta for f during tstep
  RBT, RV
Axioms
  axm01: tstep ∈ REAL_STR_POS
  axm02: max_df ∈ REAL_POS
          //  max diff  of  f  during tstep
  axm03: leq(max_df,z)
  axm04: partition(VT, {RBT}, {RV})
End
```

Model 6.7 – Extract of context *C4_discrete*

The Event-B refinement with discretization

The defined machine *M2_dsct_ctrl* (Model 6.8) produces the discrete behavior of the continuous function *fc* with the discrete function *fd* glued by invariant *glue01*. The other invariants *inv01* and *inv02* preserve *Req.2.2* and *inv03* states that the elapsed time *et* is less than the discrete time *tstep*. According to Figure 6.3, three events for `ROT`, `RBT` and `RFT` are defined, refining the `run` event. The `run_from_tick` (`RFT`) event starts the computation between two consecutive discrete values of function *fd* and fixes an arbitrary value of the variant `rs`.

The most interesting part in this machine relates to the `run_between_tick` (`RBT`) event which shall avoid the Zeno problem. For this purpose, each time this event is active, it triggers the event `run_variant` which decreases the variant. Once, this variant reaches the value 0, the `run_on_tick` (`ROT`) event is triggered to compute the final value corresponding to next discrete value of the function *fd*.

Note that the guard *grd15* is fundamental to guarantee that values are not out of the safety corridor. This assumption results from the physical plant definition.

Implementation The machine *M2_dsct_ctrl* could be used as the basis for a concrete implementation where only discrete variables (such that *i* and *fd*) would be considered and where only the event `run_on_tick` would be used to generate code.

```
Machine M2_dsct_ctrl
Refines M1_cntn_ctrl Sees C3_cast, C4_discrete
Variables
  fv,
  active,
  fc,
  now,
  active_t,
  fd // discrete power function
  i // the current instant number
  et // time elapsed from previous discrete value sampling time
  rs // remaining continuous micro steps inside the discrete macro step
  nv // next variant-related event type
Invariants
  type01: fd ∈ 0..i →REAL_POS
  type02: i ∈ ℕ
  type03: et ∈ REAL_POS
  type04: rs ∈ ℕ
  type05: nv ∈ VT
  glue01: ∀ n · n ∈ 0..i ⇒fc(cast(n) mult tstep) = fd(n)
                                                    // n ∈ 0..i ⇒fc(n∗tstep) = fd(n)
  glue02: now = (cast(i) mult tstep) plus et // now = i∗tstep + et
  inv01: ∀ n · n ∈ 0..i−1 ⇒(
             ∀ t · (leq(cast(n) mult tstep , t)
                    ∧ leq(t , cast(n+1) mult tstep))
                 ⇒(leq(fd(n) sub max_df , fc(t))
                    ∧ leq(fc(t) , fd(n) plus max_df)))
  inv02: ∀ t · (leq(cast(i) mult tstep , t) ∧ leq(t , now)) ⇒(
               leq(fd(i) sub max_df , fc(t)) ∧ leq(fc(t) , fd(i) plus max_df))
  inv03: smr(et,tstep)
Variant
  rs
Events
  Event run_from_tick ≙Refines run
    Any new_fv, dt, nfc
    Where
      ...
      grd13: et = zero
      grd14: smr(dt , tstep)
      grd15: ∀t · t ∈ dom(nfc) ⇒
               leq(fd(i) sub max_df , nfc(t))
               ∧ leq(nfc(t) , fd(i) plus max_df) // physical assumption
    Then
      ...
      act04: et := et plus dt
      act05: rs :∈ ℕ
      act06: nv := RBT
    End
```

```
 Event run_between_ticks ≙Refines run
   Any new_fv, dt, nfc
   Where
     ...
     grd13: smr(zero, et)
     grd14: smr(et plus dt , tstep)
     grd15: ∀t · t ∈ dom(nfc) ⇒
                 leq(fd(i) sub max_df , nfc(t))
                 ∧ leq(nfc(t) , fd(i) plus max_df)
     grd16: nv = RBT
     grd17: rs > 0
   Then
     ...
     act04: et := et plus dt
     act05: nv := RV
   End
 Event run_variant ≙
   Where
     grd01: nv = RV
     grd02: rs > 0
   Then
     act01: rs :| rs' ∈ ℕ ∧ rs' < rs
     act02: nv := RBT
   End
 Event run_on_tick ≙Refines run
   Any new_fv, dt, nfc
   Where
     ...
     grd13: et plus dt = tstep
     grd14: smr(zero,et)
     grd15: ∀t · t ∈ dom(nfc) ⇒
                 leq(fd(i) sub max_df , nfc(t))
                 ∧ leq(nfc(t) , fd(i) plus max_df)
    grd16: rs = 0
   Theorems
     thm03: cast(i+1) mult tstep = now plus dt
   Then
     ...
     act04: i := i + 1
     act05: fd(i+1) := new_f
     act06: et := zero
   End
End
```

Model 6.8 – Extract of machine *M2_dsct_ctrl*

6.4.4 Proofs statistics

All these models have been formalized using the Rodin Platform. As shown on Table 6.4, the main machine and the refinements led to 265 proof obligations. 67 were proven automatically and 198 needed numerous interactive proof steps.

The interactive proofs mainly relate to the use of the Theory plug-in to handle reals. The lack of dedicated heuristics due to the representation of reals as an abstract data type, and not as a native type led to more interactive proofs.

Table 6.4 – Rodin proofs statistics

Event-B model	Automated proofs	Interactive proofs	Total
C0_reals	1	29	30
C1_corridor	0	6	6
C2_margin	0	10	10
C3_cast	11	26	37
C4_discrete	0	1	1
M0_spec (top-level)	11	6	17
M1_cntn_ctrl (1st ref.)	22	51	73
M2_dsct_ctrl (2nd ref.)	22	67	89
Total	67	198	265

The sizes of the various proofs for the various machines and contexts are depicted in Figure 6.5.

In our development we use mathematical reals. We do not use floating-point numbers, they may be introduced in further refinements which is out of the scope of our work. So, we are not exploiting the results from automated verification tools on floating-point numbers [Mul+10]. Static analysis [Gou01] or abstract interpretation [CC77] (with tools such as Astrée [Cou+05]) have proved very powerful to analyze such programs. Our approach remains at a modeling level. Moreover, the set of axioms for reals in the Theory plug-in we have used does not define reals in a constructive manner. So, we were not able to use the results obtained by the Coq [BLM15] advanced proof tactics on reals. Indeed, our proofs have been discharged using the interactive prover of Rodin, leading to a large proof effort.

6.5 Conclusion

The development of cyber-physical systems requires to handle the behavior of the physical plant (environment). This behavior is usually defined using continuous time and is thus described by continuous functions producing feedback information to the controller, which in turns produces orders to the actuators. In this chapter, we have shown that it is possible to compose the development of both a controller and the corresponding behavior of the physical plant. The controller corresponds to a hybrid automaton. A simple one, with a single controlled variable, has been

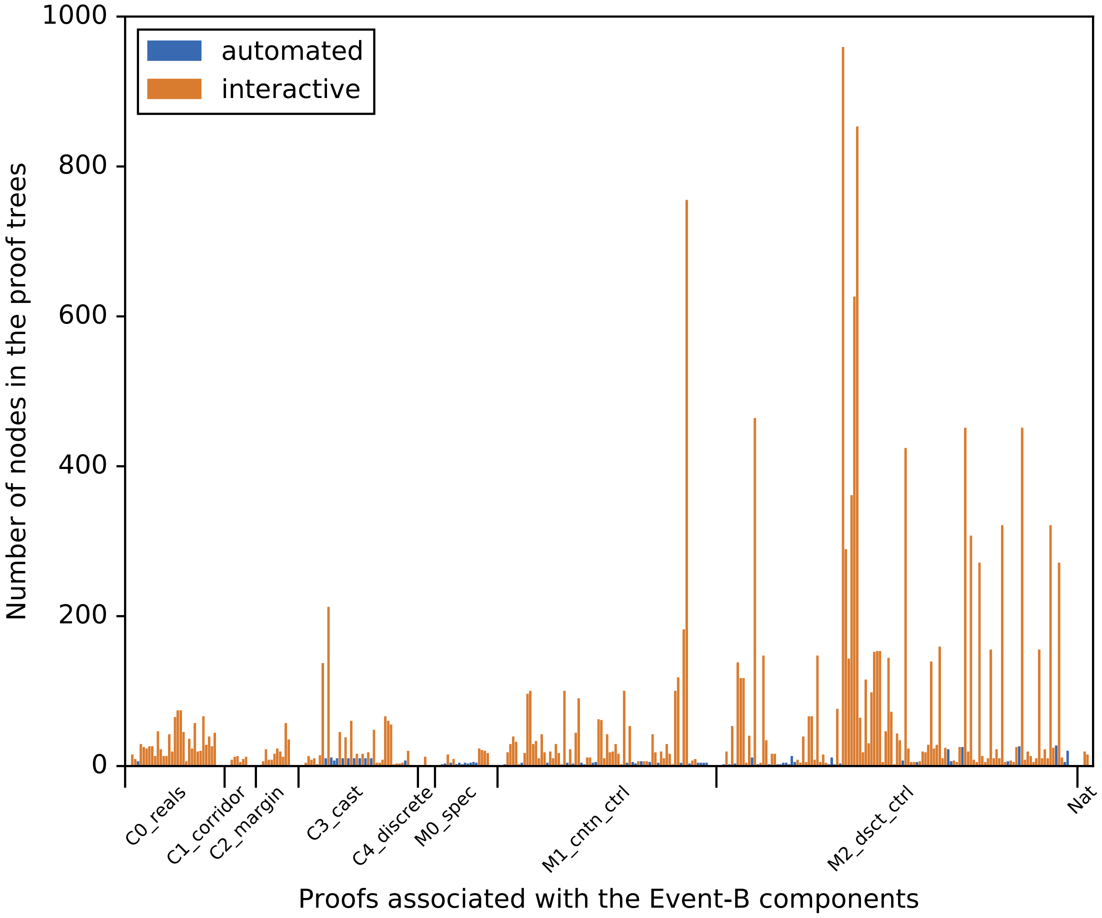


Figure 6.5 – Proofs size (number of nodes in the proof trees)

considered in this chapter. It consists in booting, running and then stopping a physical plant (see Figure 6.1).

The main contribution concerns the synthesis of a discrete controller. We have shown that the synthesis of a correct-by-construction discretization of a continuous function associated to the behavior of a physical plant can be obtained by refinement. The proof of the preservation of the invariants gluing the continuous and discrete levels guarantees this correctness. We have introduced at the discrete level a variant guaranteeing that the model is Zeno-free. The Theory plug-in for the Rodin Platform and a theory of real numbers have been used to model continuous functions. To the best of our knowledge, this is the first attempt to model continuous controller discretization with the Event-B method and mathematical reals with Rodin. This work has been published in [Bab+15].

In the next chapter (Chapter 7), we show how the substitution framework presented in Chapter 4 is set up to model the substitution of continuous systems introduced in this chapter.

7 Hybrid systems: Substitution

Chapter organization. Section 7.2 explores an incremental proof-based formal development of system substitution for hybrid systems. Finally, Section 7.4 concludes the chapter with some future research directions.

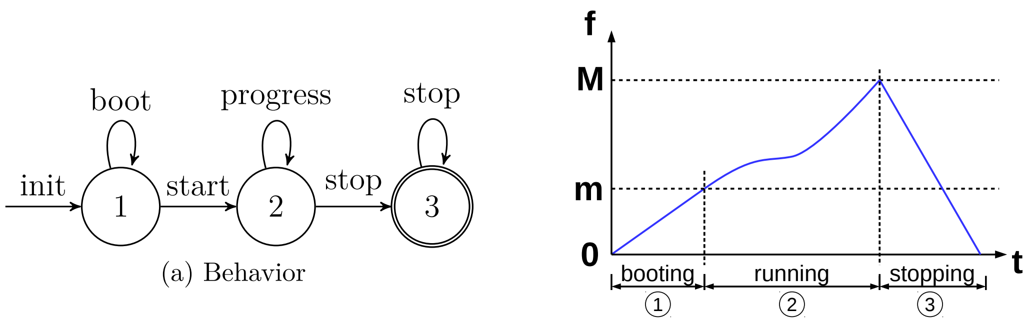
7.1 Introduction

In previous chapters, we proposed the development of a system substitution mechanism (Chapters 4 & 5) and the development of discrete controllers derived from continuous ones (Chapter 6). More precisely, we defined the reconfiguration mechanism to maintain a safety property for a system (defined as a state-transition system) during failure by switching from one supporting system to another. The defined approach has been successfully applied, for the discrete case, on web services (Chapter 5). But it is not applicable straightforwardly for hybrid systems which need to handle continuous features. In Chapter 6, we presented the formal development of a continuous controller that is refined by a discrete controller preserving the continuous functional behavior and the required safety properties. This work helped us formulate more general strategies, introduced in this chapter, for the development of system substitution for hybrid systems using formal techniques.

Hybrid systems are dynamic systems that combine continuous and discrete behaviors to model complex critical systems, such as avionics, medical, and automotive, where an error or a failure can lead to grave consequences. For critical systems, recovering from any software failure state and correcting the system behavior at runtime is mandatory. Our system substitution mechanism is an approach that can be used to recover from failure by replacing the failed system.

Objective of this chapter. Our prime objective is to model hybrid systems, and to provide modeling patterns for reconfiguration, using a correct-by-construction approach. This chapter contributes to setting up a novel technique for formalization and verification of a generic system substitution mechanism for hybrid systems that allows a system to be maintained in a safety envelope after failure by switching from one supporting system to another. We use stepwise refinement in Event-B. Moreover, we also show how the defined substitution or reconfiguration mechanism allows handling hybrid systems characterized by continuous functions and continuous time. We use the results of the previous chapter with discrete functions to address the problem of modeling the continuous systems in discrete form while preserving the continuous behavior. Particularly for hybrid systems, the system substitution is not instantaneous, and it takes time to restore the state of the substituted system. In fact, we require special treatment to handle it. The primary use of the models is to assist in the construction, clarification, and validation of the continuous controller requirements to build a digital controller in case of system reconfiguration or system substitution. In this development, we use the Rodin Platform to manage model development, refinement, proofs checking, verification and validation.

Reminder. As detailed in Section 3.2.3, we want to combine two systems whose behavior and output are represented by Figure 7.1 in order to obtain a global system whose behavior and output are modeled by Figure 7.2 and is able to substitute a system by another one in case of failure.

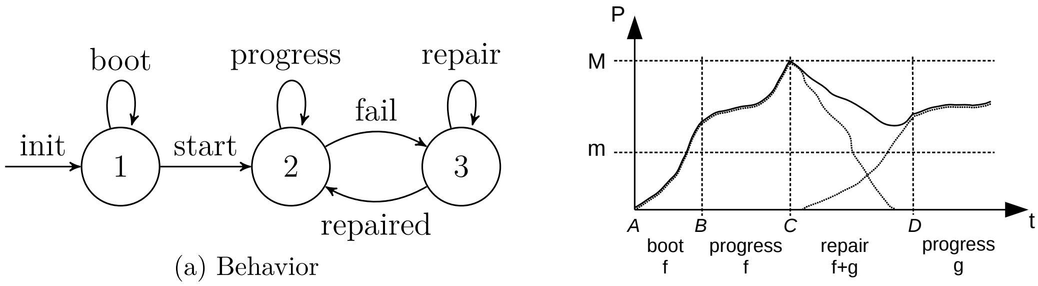


(a) Behavior

(b) Output

Figure 7.1 – Single system behavior and output

The studied systems are formalized as state-transition systems. The behavior of such systems is characterized by three states: *boot* (1), *progress* (2) and *stop* (3). The *boot* state is known as initial state, and the *progress* state is known as nominal state of studied systems. According to Figure 7.1a, after initialization, a system enters into the *booting* state, denoted as *state* 1, which may take a certain amount of time. If a system does not require the booting phase, then the system initialization



(a) Behavior

(b) Output

Figure 7.2 – Global system behavior and output

is followed by the *start* transition without any delay. After the *start* transition, the system moves into the *progress* state, denoted as *state* 2, known as running state. If the system stops, it switches into the *stop* state that is denoted as *state* 3.

7.2 Formal development

In this chapter, we model the system defined in Section 3.2.

This section describes the stepwise formal development of the systems selected for our pattern of system behavior, composed of an abstract model and a sequence of refined models. The abstract model formalizes only the system's basic behavior, while the refined models are used to define the concrete and more complex behaviors in a progressive manner that preserves the required safety properties at every refinement level.

Complete formal models are available in Appendix D.

7.2.1 The required contexts

The context *C_reals* (already presented in Model 6.1 page 86) defines the positive mathematical real numbers and theorems helpful for discharging the proofs.

Model 7.1 introduces the constants defining the different system modes: *MODE_F*, *MODE_G* and *MODE_R* for Sys_f, Sys_g and *Repair* modes) belonging to the *MODES* set.

The next two contexts (*C_envelope* and *C_margin*) deal with the definition of the safety envelope. As mentioned in the requirements defined in Table 3.1, we define the interval of safe values in $[m, M]$ in the continuous case and in $[m + z, M - z]$ with margin z in the discrete case.

```
Context C_modes
Sets
  MODES
Constants
  MODE_F, MODE_R, MODE_G
Axioms
  axm1: partition(MODES, {MODE_F}, {MODE_R}, {MODE_G})
End
```

Model 7.1 – Modes definition

```
Context C_envelope  // Safety envelope
Extends C_reals
Constants
  m, M
Axioms
  axm01: m ∈ REAL_STR_POS
  axm02: M ∈ REAL_STR_POS
  axm03: smr(m,M)
Theorems
  thm01: m ≤ M
  thm02: 0 ≤ m
  thm06: 0 ≤ M
  thm03: ∀x · m ≤ x ⇒x ∈ REAL_POS
  thm05: ∀a · m ≤ a ⇒0 ≤ a
End
```

Model 7.2 – Context *C_envelope*

```
Context C_margin  // Safety envelope margin
Extends C_envelope
Constants
  z
Axioms
  axm01: z ∈ REAL_POS // z ∈ R+
  axm02: M−m > 2∗z
Theorems
  thm03: 0 ≤ M−z
  thm06: z ≤ M−m
  thm07: m ≤ M−z
  thm08: m+z ≤ M
  thm10: m+z ≤ M−z
 ...
End
```

Model 7.3 – Context *C_margin*

7.2.2 Abstract model: definition of a mode controller

As shown in Figure 7.2a, we use three states to define a simple abstract controller (a mode automaton) that models system substitution through mode changes. Machine *M0* (see Model 7.4) describes the abstract specification corresponding to the reconfiguration state-transition system depicted in Figure 7.2a. The modes are used in the guards of events to switch from one state to another. At initialization, Sys_f is started ($MODE\_F$), it becomes active when the *active* variable is true (Sys_f ended the booting phase). When a failure or a halting condition occurs, progress of Sys_f is stopped. The controller enters in the repairing mode $MODE\_R$. Once the system is repaired, the mode is switched to $MODE\_G$ and Sys_g enters into the progress state.

```
Machine   M0
Sees   C_modes
Variables
  active // true the system is started
  md     // running mode of the system
Invariants
  type01: active ∈ BOOL
  type03: md ∈ MODES
  tech01: active = FALSE
                ⇒md = MODE_F
Events
  Event Initialisation ≙
    Begin
      act1: active := FALSE
      act2: md := MODE_F
    End
  Event boot ≙
    Where
      grd1: active = FALSE
      grd2: md = MODE_F
    End
  Event start ≙
    Where
      grd1: active = FALSE
      grd2: md = MODE_F
    Then
      act1: active := TRUE
    End
  Event progress ≙
    Where
      grd2: active = TRUE
      grd1: md = MODE_F
              ∨ md = MODE_G
    End
  Event fail ≙
    Where
      grd2: active = TRUE
      grd1: md = MODE_F
    Then
      act1: md := MODE_R
    End
  Event repair ≙
    Where
      grd2: active = TRUE
      grd1: md = MODE_R
    End
  Event repaired ≙
    Where
      grd2: active = TRUE
      grd1: md = MODE_R
    Then
      act1: md := MODE_G
    End
End
```

Model 7.4 – The mode automaton

7.2.3 First refinement: introduction of the safety envelope

The first refinement introduces the safety envelope $[m, M]$ representing the main invariant property fulfilled by all the functions f, $f+g$ during substitution and g after substitution. Machine *M1*, defined in Model 7.5, refines *M0*. It preserves the behavior defined in *M0* and introduces two kinds of events [SAZ14]:

- environment events (event name prefixed with `ENV`): they produce the system feedback observed by the controller. In this refinement, three new real variables f, g and p are introduced. The variables f and g record the feedback information of Sys_f and Sys_g individually, while p records the feedback information of the global system before, during and after substitution. The variable p corresponds to f of Sys_f in *MODE_F*, g of Sys_g in *MODE_G* and $f+g$ of combined Sys_f and Sys_g in *MODE_R* corresponding to the system repair (invariants *mode01* to *mode05*). In all cases, p shall belong to the safety envelope (invariants *envelope01* and *envelope02*).

  The `ENV` events observe real values corresponding to the different situations where Sys_f and Sys_g are running or when Sys_f fails and Sys_g boots. This last situation corresponds to the repair case.

- controller events (event name prefixed with `CTRL`): they correspond to refinements of the abstract events of *M0*. They modify the control variable *active* and *md*.

```
Machine M1 Refines M0
Sees C_envelope, C_modes
Variables
    active, md, p, f, g
Invariants
     ...
    envelope01: p ≤ M
    envelope02: active = TRUE ⇒m ≤ p

    mode01: md = MODE_F ⇒p = f
    mode04: md = MODE_F ⇒g = 0
    mode02: md = MODE_R ⇒p = f + g
    mode03: md = MODE_G ⇒p = g
    mode05: md = MODE_G ⇒f = 0
Theorems
    ...
Events
  Event Initialisation ≙
     ...
  Event CTRL_started Refines start ≙
    Where
        grd3: m ≤ p ∧p ≤ M
    End
```

```
Event ENV_evolution_f Refines progress ≙
  Any  new_f
  Where
      grd2: active = TRUE ∧md = MODE_F
      grd5: f ≠ m ∧ f ≠ M
      grd3: m ≤ new_f
      grd4: new_f ≤ M
  Then
      act1: f := new_f
      act2: p := new_f
  End

Event CTRL_limit_detected_f Refines fail ≙
  Where
      grd5: f = m ∨f = M
  End

Event ENV_evolution_fg Refines repair ≙
  Any new_f, new_g
  Where
      grd3: m ≤ new_f + new_g
      grd4: new_f + new_g ≤ M
      grd5: 0 ≤ new_f
      grd6: new_f ≤ f
      grd7: g ≤ new_g
      grd8: new_g ≤ M
  Then
      act1: f := new_f
      act2: g := new_g
      act3: p := new_f + new_g
  End

Event CTRL_repaired_g Refines repaired ≙
  Where
      grd3: m ≤ g
      grd4: g ≤ M
      grd5: f = 0   // f+g to g is continuous
  End

Event ENV_evolution_g Refines progress ≙
    ...
End
```

Model 7.5 – Refinement with `ENV` and `CTRL` events

7.2.4 Second refinement: continuous behavior and continuous time

We introduce a continuous controller defined on continuous time which characterizes its behaviors with continuous functions. It is described in Machine *M2* (see Model 7.6). It models the behavior corresponding to Figure 7.3a. Once the modes and the observed values are correctly set, the next refinements are straightforward. They correspond to a direct reuse of the development of a correct discretization of a continuous function as realized in Chapter 6. Indeed, continuous functions f_c, g_c, p_c and md_c corresponding to the variables f, g, p and md in *M1* are introduced. A real positive variable now is defined to represent the current time. The gluing invariants (for example $glue01$: $p = p_c(now)$) connect the variables of machine *M1* with the continuous functions values at time now in *M2*.

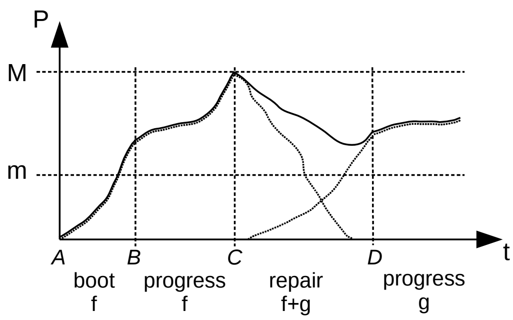
```
Machine M2 Refines M1
Sees C_corridor, C_thms
Variables
    now, p_c, f_c, g_c
    ...
Invariants
    type01: now ∈ REAL_POS
    glue01: p = p_c(now)
    glue02: f = f_c(now)
    glue03: g = g_c(now)
    corridor01: ∀t · t ∈ [0,now] ⇒p_c(t) ≤ M
    ...
Events
 ...
  Event ENV_evolution_f
    Refines ENV_evolution_f ≙
    Any dt, new_f_c
    Where
        ...
      grd5: f_c(now) = new_f_c(now)
      grd6: ∀ t · t ∈ [now,now+dt] ⇒new_f_c(t) ∈ [m,M]
    With
        new_f: new_f = new_f_c(now + dt)
    Then
     act1: now := now + dt
     act2: p_c := p_c ⩤new_f_c
     act3: f_c := f_c ⩤new_f_c
      ...
  End
 ...
End
```

Model 7.6 – Machine *M2*

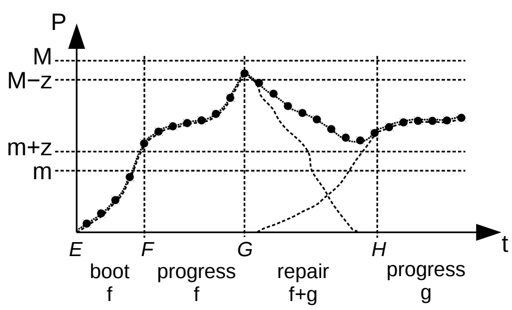
In the same way, each event of *M1* is refined. Time steps dt are introduced and the continuous functions are updated by the environment `ENV` events. The continuous functions are updated on the interval $[now, now + dt]$ and now with $now := now + dt$. The control `CTRL` events observe the value $p_c(now)$ to decide whether specific actions on the mode md_c variable are to be performed or not. Model 7.6 shows an extract of this machine and the detailed description of this refinement is given in Chapter 6.

7.2.5 Third refinement: discretization of the continuous behavior

This last refinement models a discrete controller. A discrete function is associated to values of the continuous function at each discrete time steps. The discrete behavior is described in Machine *M3* (see Model 7.7). It models the behavior corresponding to Figure 7.3b. Here again, we follow the same approach as for the refinement of the continuous behavior. As mentioned in the context C_margin, the margin z is defined, such that $0 < z \wedge m + z < M - z \ \wedge \ M - m > 2 \times z$. This margin defines, at the discrete level, the new safety envelope $[m + z, M - z] \subset [m, M]$. The new discrete variables f_d, g_d, p_d and md_d of *M3* are glued to f_c, g_c, p_c and md_c of *M2*. They correspond to discrete observations feedback of f_c, g_c, p_c and md_c. The discretization step is defined as δt. Each environment event corresponding to a continuous event is refined into three events following our strategy presented in Chapter 6. The discrete controller only observes the events on time jumps *i.e.* at instants $n \times \delta t$.



(a) Continuous system substitution



(b) Discrete system substitution

Figure 7.3 – Continuous and discrete system substitution

Note that due to the discretization and to the introduction of the z margin, a possible failure can be detected when $p_d(now) \in [m, m+z[\vee p_d(now) \in]M - z, M]$. The predicted behavior is enforced by the discrete controller that detects a limit before the value of m or M is reached. This situation is depicted in Figure 7.3b at instant G.

```
Machine M3 Refines M2
Sees C_discrete, ...
Variables
    p_d, f_d, g_d
    i  // the current instant number
    et // time elapsed from previous discrete value sampling time
     ...
Invariants
    type01: p_d ∈ 0..i →REAL_POS
    type02: i ∈ ℕ
    glue01: ∀ n· n ∈ 0..i ⇒p_c(n∗tstep)=p_d(n)
    glue02: now = i∗tstep + et
 ...
Events
 ...
  Event ENV_evolution_f_on_tick
    Refines ENV_evolution_f ≙
    Any dt, new_f_c
    Where
        ...
    Then
        act01: f := new_f
        act02: now := now + dt
        act03: f_c := f_c ⊲-new_f_c
        act04: i := i + 1
        act05: f_d(i+1) := new_f_c
        act06: et := 0
         ...
    End
 ...
End
```

Model 7.7 – Machine *M3*

7.3 Proof effort

Table 7.1 shows the proof statistics of the development with the Rodin Platform. To guarantee the correctness of the system behavior, we established various invariants in the incremental refinements. This development resulted in 732 (100%) proof obligations, of which 202 (28%) were proven automatically, and the remaining 530 (72%) were proven interactively using the Rodin prover (see Table 7.1). These interactive proof obligations are mainly related to the complex mathematical expressions and the use of *Theory* plug-in for `REAL` datatype (*i.e.* the mathematical real numbers), which are simplified through interaction, providing additional information for assisting the Rodin prover.

We use the *Theory* plug-in for describing the hybrid systems and the required properties. In this experiment, we found that proofs are quite complex and the existing Rodin tool support is not powerful enough to prove the generated proof

Table 7.1 – Proof statistics

Model	**Total number of POs**	**Automated proofs**	**Interactive proofs**
Abstract model (M0)	5	5 (100%)	0 (0%)
First refinement (M1)	93	48 (52%)	45 (48%)
Second refinement (M2)	209	71 (34%)	138 (66%)
Third refinement (M3) [projections]	425	78 (18%)	347 (82%)
Total	**732**	**202 (28%)**	**530 (72%)**

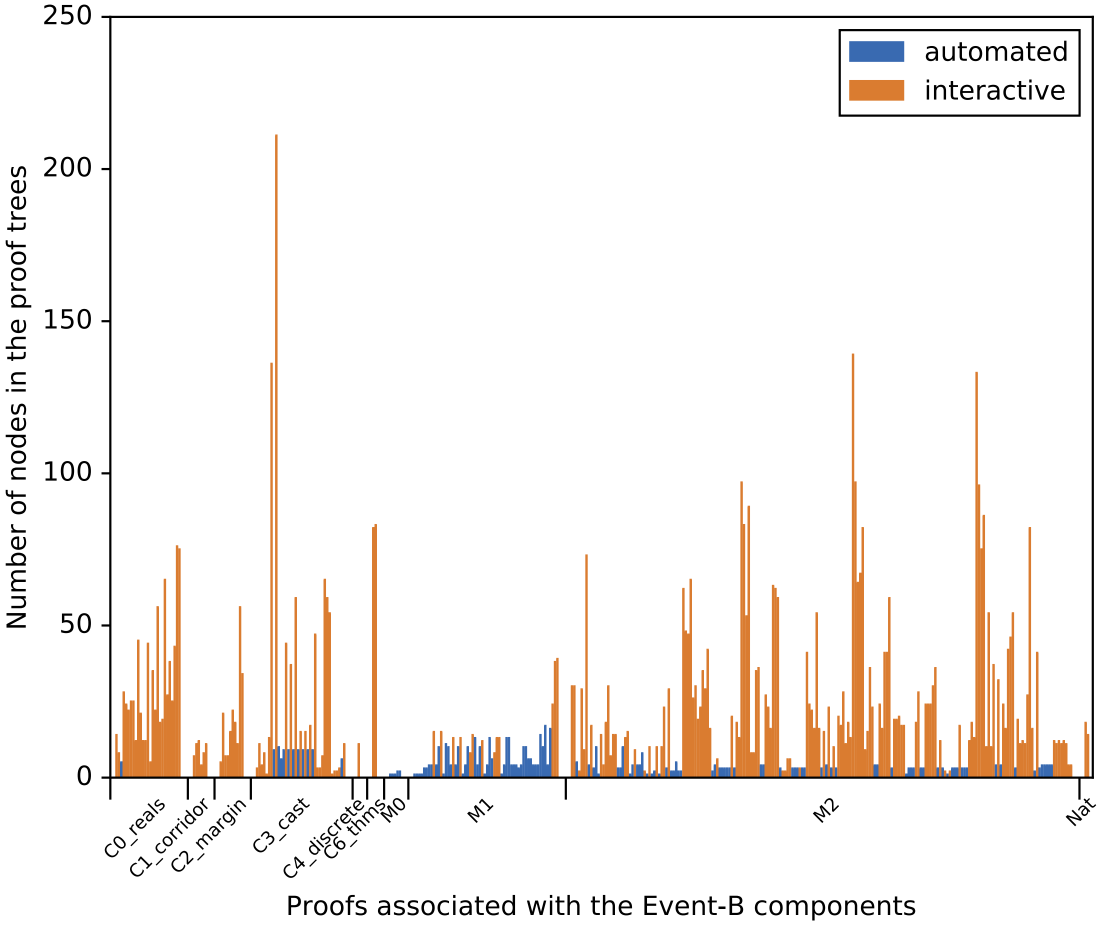


Figure 7.4 – Proofs size (number of nodes in the proof trees)

obligations automatically. In fact, we need to assist the Rodin provers in finding the required assumptions and predicates to discharge the generated proof obligations. On the other hand, we also found that the *Theory* plug-in is not yet complete. This work was done using Rodin 2.8, the *Theory* plug-in 2.0.2 and the *Real* theory from the Standard Library 0.1. In order to discharge successfully the proof obligations, we had to define several theorems, some of them as axioms, so as not to prove basic mathematical properties on reals.

The sizes of the various proofs for the various machines and contexts are available in Figure 7.4.

7.4 Conclusion

In this chapter, we have used our existing approaches for addressing the challenges related to formal modeling and verification for the system substitution for hybrid systems. This work is a preliminary step for applying the system substitution mechanism for hybrid systems. It has been published in [Bab+16b] and [Bab+16a].

We identified the following development steps to integrate the system substitution mechanism for hybrid systems:

1. Define a set of modes for the controller;

2. Define a safety envelope to preserve the desired behavior;

3. Handle the continuous behavior and continuous time;

4. Model the discretization of the continuous function.

Use of system substitution mechanisms for hybrid systems is a challenging problem as it requires to maintain a safety envelope through discrete implementation of continuous functions. To address this problem, we have presented a refinement-based formal modeling and verification of system reconfiguration or substitution for hybrid systems by proving the preservation of the required safety envelope during the system substitution process. In this chapter, we have extended the work of Chapter 5 on system substitution to handle systems characterized by continuous models. First, we formalized the system substitution at continuous level, then we developed a discrete model through refinement by preserving the original continuous behavior. The whole approach is supported by proofs and refinements based on the Event-B method. Refinements proved useful to build a stepwise development which allowed us to gradually handle the requirements. Moreover, the availability of a theory of mathematical real numbers allowed us to introduce continuous behaviors which usually rise from the description of the physics of the controlled plants. All the models have been encoded within the Rodin Platform. These developments required many interactive proofs in particular after the introduction of real numbers. The interactive proofs mainly relate to the use of the *Theory* plug-in for handling mathematical real numbers. Up to our understanding, the lack of dedicated heuristics due to the representation of real numbers as an axiomatically defined abstract data type, and not as a native Event-B type together with our limited experience in defining tactics led to this number of interactive proofs.

After showing how our proposed substitution mechanism applies to both discrete and continuous systems, we address, in the next chapter (Chapter 8), the generalization of our framework.

8 Generalization

Chapter organization. The mathematical setting that describes the generalization of the approach is presented in Section 8.2. Next, the corresponding Event-B models handling this generalized model are described in Section 8.3 and the associated instantiation mechanism is explained in Section 8.4. An example is used to instantiate this generic model in Section 8.5. Then, an assessment of the proposed approach is shown in Section 8.6, and finally, a conclusion summarizes our contribution and some future research paths are discussed in the last section.

8.1 Introduction

In this chapter we propose a generalization of our substitution framework introduced in Chapter 4. In order to demonstrate it, we will instantiate it on the discrete case already presented in Chapter 5 and obtain a similar final refined model.

Objective of this chapter. This chapter proposes a generic system reconfiguration formal model developed using correct-by-construction stepwise refinement and proof-based formal methods. Event-B supports the whole formal development of the system substitution operator. The developed generic model can be instantiated to any number of systems to be substituted. The proposed approach is generic: it depends on neither the internals of the systems nor the type of repair. An instantiation mechanism, based on a specific refinement with witnesses, is proposed to overcome the state space explosion problem usually encountered when model checking-based verification techniques are set up.

Every time a substitution case needs to be considered, we have to perform a complete formal development in order to apply the approach detailed in the previous chapters. In this sense, the previous approach provides a correct substitution mechanism, but it is not generic. Neither the development nor the verification processes can be reused. Instead of applying the previously described development for every system, we advocate the use of a generic correct-by-construction approach. The proposed generalization consists in expressing the system elements as first-order objects manipulated by the Event-B models and then building specific systems as instances of these objects. Systems, states, transitions, invariants, variants, *etc.* become objects of the proposed model, and the described system behavior conforms to Figure 2.4 page 25.

8.2 Mathematical setting for substitution

The formal mathematical setting to handle the system substitution is given below, providing the basic mathematical definitions to characterize systems. All the elements describing systems and their behavior are introduced: variables, states, variants, invariant, events and systems

8.2.1 Variables and states

Variables represent states. They belong to a set *Variables*. Their values are taken in the set *ValueElements*. Variables are associated to their values by a partial function, called *valuation*, belonging to the set *Valuations*, defined as:

$$Valuations \subseteq Variables \rightarrow\!\!\!\!\!+ \mathbb{P}(ValueElements)$$

8.2.2 Systems

Systems belong to the set *Systems* of all the systems. A system is a tuple defined as a structure involving all the features composing a system. So, for all *system* in *Systems*, we define

$$system = \left\langle variables, variant, invariant, init, progress \right\rangle$$

where:

- *variables* is a set of variables representing the state of the system:

$$variables \subseteq Variables$$

- *variant* is a function producing the natural value of the variant from a valuation of the variables:

$$variant \in Valuations \rightarrow \mathbb{N}$$

- *invariant* is a predicate defined on the variables values:

$$invariant \in Valuations \rightarrow \text{BOOL}$$

- *init* and *progress* are two generic before-after predicates recording state changes.

8.2.3 Initialization and progress

The initialization of the global system selects the first system to run. The `progress` event models a trace of assignments of new valuations for the system state variables that satisfy the invariant.

8.2.4 Systems substitution relation

System substitution requires the definition of a relation associating the source system states with the target system ones. As defined in Equation (8.1), this relation is given by the definition of an invariant, named *horizontal invariant*, as defined in previous chapters (see Section 4.3.3).

$$\boxed{\begin{array}{l} \forall S_S, S_T \in Systems. \\ \forall Inv_H(S_S, S_T) \in states(S_S) \times states(S_T) \rightarrow \text{BOOL}. \\ substitute\_states(S_S, S_T) = \\ \qquad\qquad \{(s_S, s_T) \in states(S_S) \times states(S_T) \mid Inv_H(S_S, S_T)(s_S, s_T)\} \end{array}} \quad (8.1)$$

Here:[1]

[1] If E is a set, then E^2 denotes the Cartesian product $E \times E$

- *states* is a function returning the possible valuations of a given system:

$$states \in System \rightarrow Valuations$$

- Inv_H is a predicate defining the horizontal invariant involving the values of the variables of the source and target systems:

$$Inv_H \in System^2 \rightarrow Valuations^2 \rightarrow \text{BOOL}$$

The invariant Inv_H links the source and target states. It plays the role of *Recover* in the proof obligation defined in Equation (4.3) page 53. In the generic model, its definition is given by an equivalence relation. This definition entails the definition of the repair relation: $repair \in Systems^2 \times (Valuations \rightarrow \text{BOOL})^2$. It is parameterized by two predicates ψ and φ according to the definition of Section 5.2.3.

$$\boxed{\begin{array}{l} \forall S_S, S_T \in Systems. \\ \forall \psi \in states(S_S) \rightarrow \text{BOOL}. \quad \forall \varphi \in states(S_T) \rightarrow \text{BOOL}. \\ repair(S_S, S_T, \psi, \varphi) = \{(s_S, s_T) \in substitute\_states(S_S, S_T) \mid \\ \hfill Inv_S(S_S)(s_S) \wedge \psi(s_S) \Leftrightarrow Inv_S(S_T)(s_T) \wedge \varphi(s_T)\} \end{array}} \quad (8.2)$$

where $Inv_S(S_X)(s_X)$ is the value (satisfied or not) of the system invariant of the system S_X in the state s_X.

Recall. The predicates ψ and φ (both different from *False*) define different repair or substitution modes.

- $\psi = True \wedge \varphi = True$ in the case S_T is an equivalent system substitute. This is the only case addressed in this chapter;
- $\psi \neq True \wedge \varphi = True$ in the case S_T upgrades S_S;
- $\psi = True \wedge \varphi \neq True$ in the case S_T degrades S_S.

8.2.5 Substitution property

The condition to substitute a system S_S by a system S_T in the case of equivalence is given by the *repairable_equiv* predicate characterizing the set of substitute systems.

$$\boxed{repairable\_equiv(S_S) = \exists S_T \in Systems \cdot repair(S_S, S_T, True, True) \neq \varnothing} \quad (8.3)$$

According to Equation (8.2), here the predicates ψ and φ are set to *True* in Equation (8.3) to obtain the equivalence addressed in this contribution.

The generic setting

Finally, the generic system of systems setting is given by a graph characterized by the pair $SoS = \langle Systems, repair \rangle$ where $Systems$ is the set of available systems (nodes) and $repair$ is the relation among the available systems (edges). The obtained graph of systems may be constrained by additional properties. For example, a property could be that each system has at least two substitute systems. This is out of the scope of this contribution.

8.3 An Event-B model for system substitution

The mathematical setting described above has been completely formalized within the Event-B method. The complete Event-B development is available in Appendix E. This development first expresses the system substitution strategy at a higher level, and then reuses this development for each specific system substitution. The specific system is obtained by instantiation of the generic model. Instantiation is defined by a particular use of refinement. Specific systems, defining instances, are witnesses of the generic development. This formalization led to the definition of a context *C0* and of two machines *M0* and its refinement *M1*.

8.3.1 Static part: required definitions

The context *C0* (Model 8.1) implements the theory associated to the system substitution relation. It introduces all the elements describing systems as formalized previously in Section 8.2.

```
Context C0
Sets
    Variables, ValueElements
Constants
    Valuations, VariablesSets, Systems, Systems_states, system_of,
    HorizontalInvs, varval_of
Axioms
    set1: finite (Variables)
    set2: finite (ValueElements)
    type1: Valuations ⊆ Variables ⇸ℙ (ValueElements)
    type2: VariablesSets ⊆ ℙ (Variables)
    prop1: VariablesSets ≠ ∅
    prop2: ∀ v1,v2 · (v1 ∈ VariablesSets ∧ v2 ∈ VariablesSets ∧ v1 ≠ v2)
                                                        ⇒v1 ∩ v2 = ∅
```

Model 8.1 – Context *C0* containing basic definitions and properties (part 1 of 3)

Two basic sets *Variables* and *ValueElements* are defined. They represent finite sets (*set1* and *set2* axioms in Model 8.1) of possible system variables and their possible values. They are used to characterize other elements defined in the `Constants` clause of Model 8.1.

- *Valuations* defines the possible values for variables (*type1*), and
- *VariablesSets* is a non empty (*prop1*) set (*type2*) containing disjoint sets (*prop2*) of the powerset of the *Variables* set.

The following elements are introduced:

- *Systems*, *Systems_states*, *system_of* to characterize the considered systems, their states and a function which returns the system associated to an input state,
- *HorizontalInvs* the invariant to repair two systems,
- *varval_of* function which returns the variant associated to a given system.

Their properties are described in the next section.

States and systems.

State variables are manipulated by the defined recovery mechanism. *Systems* is a set (finite and non empty in *prop3* in Model 8.2) characterizing the potentially available systems involved in a substitution. As stated above, they are considered as state-transition systems. In the context *C0* (Model 8.2) systems are characterized (*type3* and *type4*) by their set of state variables together with their possible values. To identify the system a state belongs to, we have introduced the *system_of* function (*fun1*) returning the system of an input state. Being a function, *system_of* ensures that a state belongs to a single system.

Remark. Observe that transitions between states are not given in the *C0* context, they will be introduced in the machine part of this generic Event-B model.

```
type3: Systems ⊆ VariablesSets ×(Valuations ⇸ℕ)
type4: Systems_states ⊆ Systems ×Valuations
 ...
prop3: finite (Systems) ∧Systems ≠ ∅
 ...
prop5: Systems_states ≠ ∅
prop6: dom(Systems_states) = Systems
 ...
fun1: system_of = (λ syst_st ∈ System_states | prj1(sys_st)))
```

Model 8.2 – Context *C0* containing basic definitions and properties (part 2 of 3)

Systems properties: invariants and variants.

The last part of this context (Model 8.3) introduces the properties required for system substitution *i.e.* the horizontal invariant for the preservation of the global system invariant and the variant to identify the recovery state.

- The statement *type10* defines the type of the *horizontal invariant* which associates corresponding repair states in systems.
- Property *prop7* guarantees that, for every system, the domain of the valuation function is the set of variables.
- Property *prop8* ensures that this invariant is well-defined on the states to be recovered.
- The variant expression is accessed by the *fvar_of* function in *fun4*. It returns, for a given state, the function computing the value of the variant, while the *varval_of* function (of *fun5*) returns, for a given state, the value of this variant.

```
  type10: HorizontalInvs
        ∈ (Systems ×Systems) ⇸((Systems_states ×Systems_states) ⇸BOOL)
  prop7: ∀ sys_st · sys_st ∈ Systems_states
                                      ⇒dom(prj2(sys_st))= prj1(prj1(sys_st))
  prop8: ∀ s1,s2,sst1,sst2,b ·
                  ((s1↦ s2) ↦ {(sst1↦ sst2) ↦ b} ∈ HorizontalInvs )
                       ⇒( s1 = system_of(sst1) ∧s2 = system_of(sst2) )
   ...
  fun4: fvar_of = (λ syst_st ∈ System_states | prj2(prj1(sys_st)))
  fun5: varval_of = (λ syst_st ∈ System_states |
                                         fvar_of(sys_st)(prj2(sys_st))))
   ...
End
```

Model 8.3 – Context *C0* containing basic definitions and properties (part 3 of 3)

8.3.2 Dynamic part: modeling the recovery behavior

The previous context introduced the definition of systems and their states, together with the notion of *horizontal invariant* describing the repair condition to guarantee preservation of the safety system properties. The second part of our generic model defines the `Machine` part to represent the behavior and system transitions.

The refinement strategy. A first machine and two refining machines are defined to model the behavioral part of our model. This decomposition has been defined to ease the proof process. At the top level (Machine *M0*), we introduce the generic specification of the system level. We observe the running system, its failure and repair and the case of complete failure (no system available for repair). The first refinement introduces the behavior of the running system (by introducing the `progress` event) and strengthens the definition of the repairing event (`repair` event) exploiting the horizontal invariant. The definition of the obtained model conforms to the system behavior pattern depicted by the transition system of

Figure 2.4. Finally, the last refinement is devoted to the instantiation of the generic model for specific cases.

As mentioned, we identify four categories of transitions. Each category corresponds to an Event-B event in the generic Event-B models. The full model containing the four transition categories (`initialization`, `progress`, `failure` and `repair`) is obtained in two steps: a top-level machine and one single refinement. This decomposition has been defined to ease the proof process The definition of the final obtained model conforms to the system behavior pattern depicted by the transition system of Figure 2.4.

The top level specification.

The first abstract machine *M0* introduces systems without manipulating system states since system behavior is not considered yet (Models 8.4 and 8.5).

Current system and state (Model 8.4)**.** The *available_systems* and *current_system* variables define respectively all the available healthy systems for substitution and the current running system.

```
Machine M0 Sees C0
Variables
  current_system, available_systems
Invariants
  type1: available_systems ⊆ Systems
  type2: current_system ∈ Systems
```

Model 8.4 – Skeleton of machine *M0* (part 1 of 2)

The `Initialisation` event (Model 8.5)**.** It defines the set of all available systems (*act1*) and the first running system arbitrary chosen (*act2*) in *Systems*, the set of all systems.

The events describing the system life cycle (Model 8.5)**.** At this first level of modeling, only the life cycle of the systems is captured. The internal behavior of each system is not observed yet.

This machine defines system modes and the failure occurrence together with the associated repair action:

- The Repair (`repair` event) consists in switching the current running system to another one selected among the available set of systems.
- When a system fails (`fail` event), it is removed from the available systems set.
- The global system (made of all the systems) has completely failed when the set of available systems is empty (`complete_failure` event).

First refinement.

Machine *M1* of Model 8.6 refines *M0* to define the final complete generic substitution model by introducing the internal system behavior.

```
Events
  Event Initialisation ≙
    Begin
      act1: available_systems := Systems
      act2: current_system :∈ Systems
    End
  Event Fail ≙
    Any system
    Where
      grd1: system ∈ available_systems
    Then
      act1: available_systems := available_systems \{system}
    End
  Event Repair ≙
    Any next_system
    Where
      grd1: next_system ∈ available_systems
      grd2: current_system ∉ available_systems
    Then
      act1: current_system := next_system
    End
  Event Complete_failure ≙
    Where
      grd1: available_systems = ∅
    Then
      skip
    End
End
```

Model 8.5 – Skeleton of machine *M0* (part 2 of 2)

```
Machine M1 Refines M0 Sees C0
Variables
  current_system_state, available_system_states
Invariants
  type1: available_systems_states ⊆ Systems_states
  type2: current_system_state ∈ System_states
  glue1: available_systems = dom(available_system_states)
  glue2: current_system = system_of(current_system_state)
Variant
  var1:  varval(current_system_state)
```

Model 8.6 – Extract of the machine *M1* (part 1 of 4)

Strengthening the invariants in the refined machine (Model 8.6). The refined machine defines new model variables in addition to the variables of the abstraction (*available_systems* and *current_system*). These new model variables deal with system states: *current_system_state* to model the state of the running system *current_system* and *available_system_states* to define all the states of the systems in the *available_systems* set. These variables are used to describe the internal behavior of systems which remained abstract in the top machine.

Two relevant gluing invariants are introduced:

- *glue1* guarantees that the considered states are exactly those corresponding to the available systems, and
- *glue2* guarantees that the *current_state* variable corresponds to the current state of the running system *current_system*.

Finally, a variant value is associated with the current state of the running system by statement *var1*.

Unchanged events (Model 8.7). The new variables are initialized at the initial state of the running system for the *current_system_state*. `complete_failure` and `fail` events remain unchanged.

```
Events
  Event  Initialisation  ≙ ...
  Event Fail Refines Fail ≙ ...
  Event Complete_failure Refines Complete_failure ≙...
```

Model 8.7 – Extract of the machine *M1* (part 2 of 4)

Introducing system behaviors: the `progress` event (Model 8.8). The new `progress` event introduces the behavior of the current system: `progress` changes the new state valuation (*act1*) to the new value defined as *new_valuation* parameter.

```
  Event Progress ≙
    Any
      new_valuation
    Where
      grd1: current_system ∈ available_systems
      grd2: new_valuation ∈ Valuations
      grd3: dom(new_valuation) = dom(valuation_of(current_system_state))
      grd4: fvar_of(current_system_state)(new_valuation)
                                              < varval_of(current_system_state)
    Then
      act1: current_system_state :=
                          system_of(current_system_state) ↦ new_valuation
    End
```

Model 8.8 – Extract of the machine *M1* (part 3 of 4)

The guard of this event requires that the *new_valuation* parameter is a possible valuation for the variables of the current state of the running system (*grd1*, *grd2* and *grd3*). Moreover, this valuation shall decrease the value of the variant to ensure progress (*grd4*).

The `progress` event at the generic level only models the coherence of the behavior but does not model any specific of the systems. Each concrete system will be a refinement of this model, and will detail its behavior by refining this `progress` event. We do not model the concrete behavior of the systems at the generic level.

Refinement of the `repair` event to handle system behaviors (Model 8.9). The refined event `repair` switches the current system to the substitute one (*act1*) and defines the recovery state in the substitute system (*act2*). Both these elements are described in terms of the variables *new_variables*, *new_variant* and *new_valuation* which are the results of the following guards:

```
Event Repair Refines Repair ≙
  Any new_variables, new_variant, new_valuation, h_inv
  Where
    grd1: current_system ∉ available_systems
    grd2: new_variables ∈ VariablesSets
    grd3: new_variant ∈ Valuations ⇸ℕ
    grd4: new_valuation ∈ Valuations
    grd5: (new_variables ↦ new_variant) ↦ new_valuation
                                          ∈ available_systems_states
    grd6: new_variables ≠ variables_of(current_system_state)
    grd7: new_variant(new_valuation) = varval_of(current_system_state)
    grd8: h_inv =
         HorizontalInvs(current_system ↦ (new_variables ↦ new_variant))
    grd9: h_inv(current_system_state ↦
               ((new_variables ↦ new_variant) ↦ new_valuation))= TRUE
    grd10: current_system ↦ (new_variables ↦ new_variant)
                                              ∈ dom(HorizontalInvs)
  With
    next_system: next_system = new_variables ↦ new_variant
  Then
    act1: current_system := new_variables ↦ new_variant
    act2: current_system_state :=
                        (new_variables ↦ new_variant) ↦ new_valuation
  End
```

Model 8.9 – Extract of the machine *M1* (part 4 of 4)

- *grd2*: the new variables set is one of the possible variable sets (typing constraint)
- *grd3*: the new variant has the correct type (partial function of the variables which outputs a natural)

- *grd4*: the new valuation is a member of the possible valuations set (typing constraint)
- *grd5*: the new state constituted of *new_variables*, *new_variant* and *new_valuation* exists and is available (has not yet failed)
- *grd6*: the new variables are not variables of the current system, which ensures that the substitute system is different from the failed one
- *grd7*: the value of the new variant computed on the new valuation of the variables is equal to the value of the variant at the current state of the system being replaced. This means that the new system will continue the work where the previous one stopped because the variant are constructed here to model the progress of the system.
- *grd8*: the horizontal invariant corresponding to the pair of systems composed of the current system and the new system is extracted from the context in the variable *h_inv*
- *grd9*: the specific horizontal invariant *h_inv* is enforced to be true on the pair on system states. This means that the state of the new system corresponds to the state of the replaced system as defined in the horizontal invariant relation.
- *grd10*: there exists an horizontal invariant defined for the pair of systems composed of the current system and the new system

Finally, a witness (`With` clause) is provided to make explicit the substitute system giving its new state variables and variant value.

8.4 Instantiation of generic Event-B by refinement

In the previous section, we have presented a generic model for system substitution corresponding to the pattern depicted on Figure 2.4. This model is divided in two parts: one modeling systems, states, variables, variants and invariants; and a second modeling the behavior of systems and of the substitution mechanism. Instantiation consists in setting up the obtained generic model for specific systems. It is obtained after two steps, described below, corresponding to the instantiation of each modeling part.

8.4.1 Step 1. The instantiation context

First, specific values of the abstract sets defined in the context *C0* presented in Section 8.3.1 are introduced. An instantiation context *C0_instance*, extending the context *C0* (Model 8.1), is defined with concrete values for all the sets (*Variables*, *ValueElements*) and for the constants (*Valuations*, *VariablesSets*, *Systems* and *System_states*). In the case of our example, they are given in Model 8.11.

8.4.2 Step 2. Refinement and witnesses for instantiation

In order to use the concrete values defined in the context *C0_instance*, a machine *M2* refining *M1* is defined. This machine contains all the specifics of the system. The behavior of the system, previously modeled in a generic way by the event `progress`, is now detailed by events `progress_sysX_ABC` corresponding to the progress events *i.e.* transitions in the specific system *sysX*. Concrete event variables of *M2* and abstract variables of *M1* – previously defined with event parameters (`Any` clause) – are glued thanks to the use of witnesses (using the `With` clause). Model 8.10 shows an example of such an instantiation: the parameter *new_status* is instantiated in this particular case with the value *OPEN*. The guard *grd1* ensures that this event modeling the *open* transition in *sys1* is only enabled when the running system is *sys1*. The second guard *grd2* models a specific element of this transition *open*.

```
Event progress ≙
  Any
    new_status
  Then
    act1: state := new_status
  End
```

```
Event progress_sys1_open
Refines progress ≙
  Where
    grd1: current_system = sys1
    grd2: state = CLOSED
  With
    new_status = OPEN
  Then
    act1: state := OPEN
  End
```

Model 8.10 – Instantiation principle: use of refinement with witnesses

8.5 Application to the case study on web service compensation

In this section, the case study presented in Section 3.1 is developed again as an instance of the generic model of Section 8.2 following the instantiation principle of Model 8.10. It is formalized as an instance of the generic approach.

8.5.1 Step 1. The instantiation context. Application to the case study

The instantiation context *C0_instance* of Model 8.11 provides concrete values for the deferred sets of the context *C0*. All the sets corresponding to the static characterization of the systems like *Variables*, *ValueElements*, *Valuations*, *VariableSets*, *Systems_states*, *Systems* and *HorizontalInvs* are valued by set comprehensions of possible instances. They characterize specific systems corresponding to the case study of Section 3.1.

- The three variables defined by *axm1* are the cart of the first system (*C1*) and the carts of the second system (*C2a* and *C2b*).
- The values of the variables are elements from *ValueElements* which is constituted of the 5 available products *Prod1* to *Prod5*.
- The valuations are restricted to only depend on the sets of variables of the systems. This prevents incoherent functions that would depend on variables from disjoint systems.
- The sets of variables of the systems are specified explicitly by axiom *axm4*.
- The first system *Sys1* is defined in *axm5* by its variable (*C1*) and its variant $(5 - \text{card}(C1))$.
- The second system *Sys2* is defined in *axm6* by its variables (*C2*a and *C2b*) and its variant $(5 - \text{card}(C2a \cup C2b))$.
- The set of all systems is defined as composed of *Sys1* and *Sys2*.
- The fundamental axiom *axm9* defines the horizontal invariants set, which is here a singleton, describing a horizontal invariant from *Sys1* to *Sys2*: $C1 = C2a \cup C2b$. It corresponds to the repair property introduced in Section 4.2.

```
Context C0_instance Extends C0
Constants
  C1, C2a, C2b, Prod1, Prod2, Prod3, Prod4, Prod5, Sys1, Sys2
Axioms
  axm1: partition(Variables, {C1}, {C2a}, {C2b})
  axm2: partition(ValueElements, {Prod1}, {Prod2}, ... , {Prod5})
  axm3: Valuations = ({C1} →ℙ(ValueElements))
                     ∪({C2a,C2b} →ℙ(ValueElements))
  axm4: VariablesSets = {{C1},{C2a,C2b}}
  axm5: Sys1 = {C1} ↦ (λ val ·val ∈ {C1} →ℙ(ValueElements) |
                                  card(ValueElements) −card(val(C1)))
  axm6: Sys2 = {C2a,C2b} ↦ (λ val ·val ∈ {C2a,C2b} →ℙ(ValueElements) |
                      card(ValueElements) − card(val(C2a) ∪val(C2b)))
  axm7: Systems = {Sys1,Sys2}
  axm8: Systems_states = Systems ×Valuations
  axm9: HorizontalInvs = {(Sys1 ↦ Sys2)↦ (λ (sst1 ↦ sst2) ·
                           sst1 ∈{Sys1}×({C1}→ ℙ(ValueElements))
                         ∧ sst2 ∈{Sys2}×({C2a,C2b} →ℙ(ValueElements)) |
             bool(valuation_of(sst1)(C1) =
                       valuation_of(sst2)(C2a) ∪valuation_of(sst2)(C2b)))}
    ...
End
```

Model 8.11 – The instantiation context *C0_instance*

8.5.2 Step 2. Refinement and witnesses for instantiation. Application to the case study

The events of machine *M1* are refined by machine *M2* (Models 8.12 & 8.13) for instantiation according to the principle of Section 8.4.2. *M2* models the instantiated machine for the events of the case study on web service compensation defined in Section 3.1.

In this machine, the concrete variables *sys1_cart*, *sys2_cart1* and *sys2_cart2* have been defined as instantiation of the abstract variables *C1*, *C2a* and *C2b*. The invariants *glue1* and *glue2* ensure the coherence between the two abstraction levels.

In the `repair_sys1_to_sys2` event, *grd6* expresses the concrete form of the horizontal invariant which was previously specified by *h_inv*, now only visible in the witness. We can also see the connection between the abstract and the concrete variables in *grd7* and *act2*.

The `progress_sys1` event (detailled in Model 8.13) corresponds to the event `addItem_WS1` (Model 5.4) of *Sys1* (one website system). It consists in adding a product (*new_prod*) in the cart *C1* of the website $site_1$. The event is defined in terms of the concrete variables and the connection with the abstract parameters is given by the witness (as well as enforced by the invariants).

```
Event progress_sys1 Refines progress ≙
  Any new_prod
  Where
    grd1: current_system = Sys1
    grd2: Sys1 ∈ available_systems
    grd3: new_prod ∈ ValueElements
    grd4: new_prod ∉ sys1_cart
  With
    new_valuation: new_valuation = {C1 ↦ (sys1_cart ∪{new_prod})}
  Then
    act1: sys1_cart := sys1_cart ∪{new_prod}
    act2: current_system_state := Sys1 ↦ {C1 ↦ (sys1_cart ∪{new_prod})}
  End
```

Model 8.13 – The generic `progress` event for one website of machine *M2*

8.6 Assessment

The main benefit of this proposal resides in the fact that the proof of correctness for the substitution strategy is performed only once. However, this proof together with the proof of refinement are more complex as they are generic.

8.6.1 Proof statistics

Table 8.1 shows the proof statistics for the whole Event-B developments. We note that a lot of efforts are devoted to the interactive proof of the instantiation. All

```
Machine M2 Refines M1 Sees C0_instance
Variables
  available_systems, available_systems_states
  current_system, current_system_state
  sys1_cart, sys2_cart1, sys2_cart2
Invariants
  glue1: system_of(current_system_state) = Sys1 ⇒
                              valuation_of(current_system_state)(C1) = sys1_cart
  glue2: system_of(current_system_state) = Sys2 ⇒
                            valuation_of(current_system_state)(C2a) = sys2_cart1
                          ∧ valuation_of(current_system_state)(C2b) = sys2_cart2
Events
  Event Initialisation  ≙ ...
  Event failure_sys1 Refines failure  ≙ ...
  Event failure_sys2 Refines failure  ≙ ...
  Event repair_sys1_to_sys2 Refines repair ≙
    Any new_sys2_cart1, new_sys2_cart2
    Where
      grd1: new_sys2_cart1 ∈ ℙ(ValueElements)
      grd2: new_sys2_cart2 ∈ ℙ(ValueElements)
      grd3: current_system = Sys1
      grd4: Sys1 ∉ available_systems
      grd5: Sys2 ∈ available_systems
      grd6: sys1_cart = new_sys2_cart1 ∪new_sys2_cart2
      grd7: Sys2 ↦ {C2a ↦ new_sys2_cart1, C2b ↦ new_sys2_cart2} ∈
                                                   available_systems_states
    With
      h_inv: h_inv = HorizontalInvs(Sys1 ↦ Sys2)
    Then
      act1: current_system := Sys2
      act2: current_system_state := Sys2 ↦ {C2a ↦ new_sys2_cart1,
                                            C2b ↦ new_sys2_cart2}
      act3: sys2_cart1 := new_sys2_cart1
      act4: sys2_cart2 := new_sys2_cart2
    End
  Event complete_failure Refines complete_failure ≙ ...
  Event progress_sys1 Refines progress ≙ ...
  Event progress_sys2_c1 Refines progress ≙...  // detailed below
  Event progress_sys2_c2 Refines progress ≙...
End
```

Model 8.12 – The instantiation machine obtained *M2* by refinement the proof obligations associated with the formal Event-B development presented here have been proved either with the automatic provers associated in the Rodin Platform or using interactive proofs handled by the developer on the Rodin Platform as well.

The key point related to scalability concerns the instantiation of specific systems. Indeed, the development presented above is a generic one, defined at a meta-level, where the proof obligations associated to the correctness of the system substitution obtained in Section 4.3.1 act as meta-theorems.

The use of the generalized substitutions (`Any` constructs) shows that the development considers any transition system described by a template corresponding to Figure 2.4 together with the associated invariants expressed in the corresponding Event-B models.

Table 8.1 – Rodin proofs statistics

Event-B model	Generated proof obligations	Automated proofs	Interactive proofs
Context *C0*	7	5	2
Machine *M0*	5	5	0
Machine *M1*	28	22	6
Instantiation context *C0_context*	3	2	1
Instantiation machine *M2*	54	39	15
Total	97	73	24

This looks very interesting and promising because this means that the substitution mechanism pattern has only to be proved once. However, the proof is more difficult than the concrete system alone. Therefore, the choice depend on the possibility to reuse a particular substitution pattern in several development projects.

Note that model checking techniques can be applied to automatically check the correctness of the instantiation. The exploration of all the possible states is possible since the sets are defined with a finite number of values in the context *C0_instance*. However, these techniques face the state explosion problem. For instance, the difficulty of the proofs in our approach is not affected by the number of products whereas a method which would have to explicitly enumerate all the possible values of the carts would be severely limited by the huge numbers of possibilities due to combinatorics.

The sizes of the various proofs for the various machines and contexts are available in Figure 8.1.

8.6.2 Correct-by-construction formal methods

The proposed approach is a generic one. The context *C0* describes the manipulated system concepts explicitly (*systems*, *variables*, *HorizontalInvs*, *etc.*). These concepts are manipulated as first-order objects in the machines *M0* and *M1* in order to

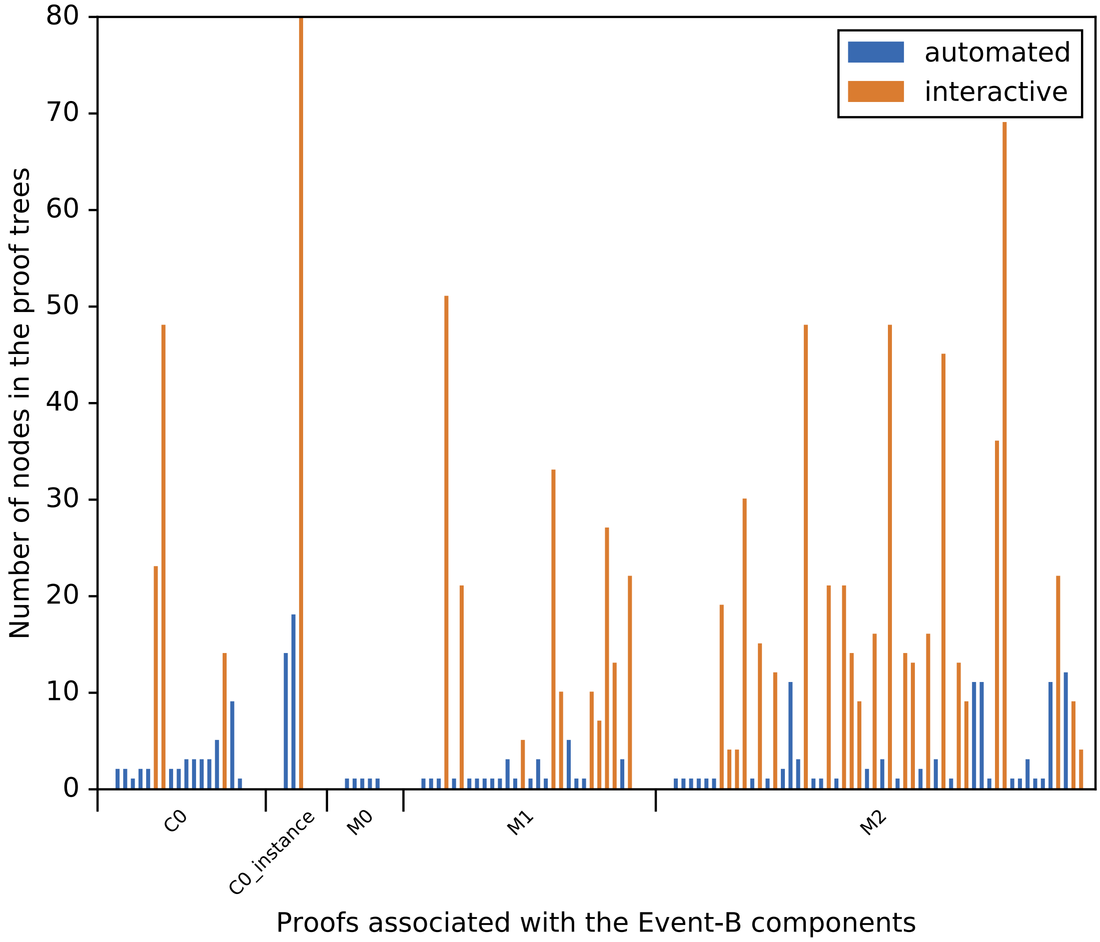


Figure 8.1 – Proofs size (number of nodes in the proof trees)

encode the behavior pattern described with the events `Initialization`, `progress`, `fail`, `repair` and `complete_failure` as show on Figure 2.4. Let us note that transitions are not manipulated as first order objects and thus not defined within the context *C0*.

One may wonder why the transitions between states are not defined explicitly in this context *C0*. There are two main reasons for that.

- First, transitions are not explicitly manipulated by the substitution mechanism we introduced. This reduces heavily the complexity of the generic model because it relies upon the refinement capabilities of Event-B to handle the modeling of the core behavior of the system.

- Second, the Event-B method provides a powerful built-in inductive proof technique based on invariant preservation by the events (see Table 1.1). This enables us to split the overall proof into smaller, more manageable proofs.

Therefore, we rely on the definition of Event-B events to define generic transitions (using the `progress` event). The proofs of invariant preservation and of variant decrease are achieved at the abstract level of machine *M1*. They are preserved by any other machine that refines it.

To instantiate these generic events for a specific system acting as a system instance, the abstract events of machine *M1* are refined. An event refining an abstract event is introduced for each concrete event of the system instance (*e.g.* the event `progress_sys1` corresponding to the concrete `addItem_WS1` event refines the abstract `progress` event). The only proof effort relates to the correct event refinement.

Note that in other traditional correct-by-construction techniques like Coq [BC04; The16] or Isabelle [NPW02; Wen16], classical inductive proof schemes are offered. One has:

- first to describe the inductive structure associated to the formalized systems,
- then to give a specific inductive proof scheme for this defined inductive structure and,
- finally to prove the correct instantiation.

In the core definition of these techniques, the inductive process associated to transition systems corresponding to the pattern of Figure 2.4 and the refinement capability are not available as a built-in inductive proof process (like in Event-B where this notion is available through state variables and events). The developer would have to formalize the notion of transition together with corresponding inductive proof principles and the instantiation of transitions because event refinement is not available.

Compared to the Event-B method, there is a need of another meta level specification and proof process.

8.7 Conclusion

In this chapter, we have presented an approach for correct system substitution that is generic and that can be instantiated to any number of systems, thus it could scale in practice. An instantiation mechanism based on the definition of witnesses has been defined. Note that, since instantiation is performed by refinement, solely the last refinement step shall be proved for each new instantiation. It corresponds to checking that the witnesses belong to the set of correct systems. From a methodological point of view, when instantiation by model checking does not scale up, one may use the defined instantiation mechanism based on witnesses. The whole proposed approach has been modeled within the Event-B method. Refinement and proof have been extensively used to obtain the whole model and its instantiations. We believe our results could be used in other formalisms because only the use of the Event-B refinement relation to link the pattern and its instantiations is specific of our tool. This work has been published in [BAP16b].

We did not apply our generic approach to systems with continuous behaviors. However, considered the work presented in the previous chapters on the modeling of the substitution in continuous systems at a concrete level, we believe that our generic approach could be applied to a continuous system.

Part III

Conclusion

Conclusion and perspectives

Conclusion

In this thesis, we addressed the problem of correct system substitution as a system development activity to handle the problem family of system evolution at design time or runtime. We consider that a source system can be substituted (replaced) by another system, namely a target system. A generic system substitution operation has been defined and formalized. Applicability of this operation on both discrete event-based systems and hybrid systems has also been demonstrated. Several contributions resulted from our work:

- First, we propose a model for a stepwise correct-by-construction method which encompasses the various characteristics of the system substitution operator we have defined. The proposed approach is based on refinement and proof and uses the Event-B method as support for the development.

  A class of systems refining a shared specification is formally developed. They represent the set of systems that may substitute each other. The designed substitution operator is parameterized by a safety property, named *horizontal invariant*, ensuring the quality of the services offered by the substitute system. This operator is able to restore the state of the source system, using this horizontal invariant, in the identified corresponding state of the target system.

  This substitution operator offers several modeling options for system substitution:

  - It can be used to replace systems at design time (when the state of the restored system is the initial state) or at runtime (when the state of the restored system is an identified state of the target system corresponding to the halting state of the source system).
  - According to the definition of the gluing invariant, this operation offers the capability to define different substitution modes: equivalent, degraded and upgraded modes.
  - When the states of the source and target systems are disjoint, the substitution corresponds to a replacement of a system by a new one. But other capabilities are offered when the halting state of the source and the restarting state of the target systems are identical (*e.g.* self-$\star$ systems,

autonomous systems, *etc.*) or when part of the source and target system states are shared (*e.g.* maintenance).

- Second, we have experimented the use of the defined system substitution operation in two situations that correspond to semantically different categories of systems where the system substitution operation was instanciated in order to handle:

  - *discrete systems* whose behavior is formalized by discrete models namely state-transition systems in our case. This use was illustrated with the web services compensation case where compensation is modeled as a service substitution. Web services compensation at runtime has been modeled as a specific definition of the proposed substitution operator. This proposal led to the definition of a new compensation mechanism for web services that is not yet formalized in the current standards of web services.
  - *hybrid systems*, or cyber-physical systems, whose behavior is continuous and require the introduction of continuous mathematical features for their modeling. We relied on the theory plug-in in Event-B in order to model these aspects.

    In general, halting and starting these systems is not instantaneous. The proposed formalization of our system substitution operator enabled us to define a system substitution on such systems. We have shown that the state restoration maintains the safety invariants even when substitution is not instantaneous, provided that some properties of the physics of the system are taken into account in the formal model.

    A formalization of the discretization of the defined continuous behaviors has been defined, it allows a developer to identify how such systems are controlled.

- Finally, we naturally studied the capability to develop the substitution operation as a generic operator that can be instantiated for any system defined as a state-transition system.

  We succeeded in generaling our approach and defined a generic model formalizing the defined substitution operator using an explicit model for states and for the horizontal invariants using lambda expressions (deep modeling) and the events of the Event-B machines to model the transitions of the considered systems (shallow modeling).

  The system substitution we defined for web services compensation has been obtained by instantiating the defined generalization. Web services and the corresponding gluing invariant has been provided as instances of the defined generalized model.

  Moreover, this generic model enabled us to concentrate the proof effort on the generalized level (reusable level of abstraction) in order to share this proof effort among several particular instantiations.

Perspectives

The results obtained in this thesis opened several new research directions. Below, we give a non-exhaustive list of the perspectives to our work.

Two types of perspectives have been identified.

The first category relates to the specific case studies of web services and cyber-physical systems modeling.

The case of web services management

- *Web services compensation.* Our model of service compensation does not make explicit the choice of the compensating service. This could be addressed using quality of service properties that may complete the functional invariants. Defining classes of services can be a solution for such a characterization. The substitute web service would be selected at runtime among the services belonging to this class.

- Several ontology models have been introduced to define semantic web services. In these ontologies, classes of functionally equivalent web services are defined and hierarchically structured using a subsumption relationship. A link between the ontology classes of target services and a given source web service could be formally established.

The cyber-physical systems

The developments we have conducted on continuous models for cyber-physical systems led to several possible extensions:

- The refinement we have defined for the discretization of continuous definitions relies on mathematical real numbers. In order to further develop our models of substitution in cyber-physical systems, it is needed to introduce another refinement from mathematical reals to floating-point numbers as another discretization step. One issue to define the gluing invariant would be to use the intermediate value theorem as gluing invariant between the discretization level with mathematical real numbers and the discretization level with floating-point level. This would enable a correct concrete implementation of the controller.

- The models defined in our work handled a single variable for information feedback (one parameter for the continuous function) with a simple safety envelope (interval that the value must belong to). Investigating an extension of the function descriptions to a set of variable parameters (vector) is needed as in traditional models in control theory. As a consequence, the safety envelope, which was defined as a simple interval, becomes a complex constraint expression denoting a constraint solving problem. More precisely, it could first be an extension of intervals to higher dimension boxes as it is done in classical interval arithmetics; but precision might require more complex relational envelopes. Proving the correctness of such models requires more powerful proof techniques.

- The other extension that needs to be studied relates to the manipulation of the continuous functions. We have used an explicit representation of a function while control theory uses differential equations to describe the continuous behaviors. We believe that our developments can manipulate function derivatives but it will also require modeling derivates and integrals using the theory plug-in and more appropriate proof techniques.

The second category of perspectives concerns the possible extensions of the defined system substitution operation:

System substitution operation

- The system substitution operation we have defined considers a fixed number of systems. One may study the case where the systems enter and/or leave the set of systems dynamically. In this case, the set of available systems evolves dynamically. This situation occurs in the case of adaptive and/or autonomic systems. In this case, the substitute system is chosen among a dynamic set of possible substitute systems and quality of service criteria may be introduced for the selection.

- Studying the formalization of the other situations like the case of self-$\star$ systems with shared variables between source and target systems, or more detailed situations for upgraded and degraded modes need to be studied in more details.

- Structuring system substitutions as relations (edges) in a graph with systems as nodes allows a designer to select which substitute systems can be used (neighbor nodes). Additionally, constraints (QoS, upgrade/degrade, *etc.*) can be added to the edges or to the whole graph (*e.g.* each node has at least three neighbor nodes). Thus, the graph expressing the substitution possibilities would be exploited for selecting target systems for substitution.

- Adding probability of failures and its corresponding calculus is an issue to address in case of safety analysis of critical systems.

- Finally, one important extension would be the substitution of a set of systems by another set of systems. The objective is to maintain an invariant for the global system (*global invariant*) corresponding to a property of an offered service while some systems composing the global system may leave or enter the global system. Each local system is characterized by its own invariant (*local invariant*). An example of such a system could be a farm of wind turbines that produce an amount of energy where some particular wind turbines may start production (windy case) or may stop (missing wind).

Studying the previously identified perspectives will certainly improve the engineering of system substitution, maintenance, reconfiguration and adaptation.

List of Publications

Guillaume Babin. “A formal approach for correct-by-construction system substitution”. In: *The Tenth European Dependable Computer Conference (EDCC) 2014 – Student Forum*. Vol. abs/1404.7513. EDCC 2014: http://arxiv.org/abs/1405.2998. Apr. 2014. URL: http://arxiv.org/abs/1404.7513.

Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. “Formal Verification of Runtime Compensation of Web Service Compositions: A Refinement and Proof Based Proposal with Event-B”. In: *2015 IEEE International Conference on Services Computing (SCC)*. June 2015, pp. 98–105. DOI: 10.1109/SCC.2015.23.

Guillaume Babin, Yamine Aït-Ameur, Shin Nakajima, and Marc Pantel. “Refinement and Proof Based Development of Systems Characterized by Continuous Functions”. In: *Dependable Software Engineering: Theories, Tools, and Applications (SETTA)*. Ed. by Xuandong Li, Zhiming Liu, and Wang Yi. Vol. 9409. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 55–70. ISBN: 978-3-319-25941-3. DOI: 10.1007/978-3-319-25942-0_4.

Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. “Correct Instantiation of a System Reconfiguration Pattern: A Proof and Refinement-Based Approach”. In: *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*. Jan. 2016, pp. 31–38. DOI: 10.1109/HASE.2016.47.

Guillaume Babin, Yamine Aït-Ameur, Neeraj Kumar Singh, and Marc Pantel. “Handling Continuous Functions in Hybrid Systems Reconfigurations: A Formal Event-B Development”. In: *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 5th International Conference, ABZ 2016, Linz, Austria, May 23-27, 2016, Proceedings*. Ed. by Michael Butler, Klaus-Dieter Schewe, Atif Mashkoor, and Miklos Biro. Springer International Publishing, 2016, pp. 290–296. ISBN: 978-3-319-33600-8. DOI: 10.1007/978-3-319-33600-8_23.

Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. “A generic model for system substitution”. In: *Trustworthy Cyber-Physical Systems Engineering*. Ed. by Alexander Romanovsky and Fuyuki Ishikawa. Computer and Information Science Series. Chapman and Hall/CRC, Sept. 2016. Chap. 4, pp. 75–103. ISBN: 9781498742450. URL: https://www.crcpress.com/Trustworthy-Cyber-Physical-Systems-Engineering/Romanovsky-Ishikawa/p/book/9781498742450.

Guillaume Babin, Yamine Aït-Ameur, Neeraj Kumar Singh, and Marc Pantel. "A System Substitution Mechanism for Hybrid Systems in Event-B". In: *Formal Methods and Software Engineering: 18th International Conference on Formal Engineering Methods, ICFEM 2016, Tokyo, Japan, November 14-18, 2016, Proceedings*. Ed. by Kazuhiro Ogata, Mark Lawford, and Shaoying Liu. Vol. 10009. Lecture Notes in Computer Science. Springer International Publishing, Nov. 2016, pp. 106–121. ISBN: 978-3-319-47845-6. DOI: 10.1007/978-3-319-47846-3_8.

Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. "Web Service Compensation at Runtime: Formal Modeling and Verification Using the Event-B Refinement and Proof Based Formal Method". In: *IEEE Transactions on Services Computing – Special Issue on Advances in Web Services Research* 10.1 (Jan. 2017), pp. 107–120. ISSN: 1939-1374. DOI: 10.1109/TSC.2016.2594782.

Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. "Correct Instantiation of a System Reconfiguration Pattern: A Proof and Refinement-Based Approach". In: *Journal of Software: Evolution and Process – Special Issue for HASE 2016 – Under revision after first review* (2017).

Guillaume Babin, Yamine Aït-Ameur, Neeraj Kumar Singh, and Marc Pantel. "Handling Continuous Functions in Hybrid Systems Reconfigurations: A Formal Event-B Development". In: *Science of Computer Programming – Special issue for ABZ 2016 – Under revision after first review* (2017).

Part IV

Appendices

A Theories

Components:

- Theory *Real* (page 140)
- Theory *RealPos* (page 145)

The models are also available at: `http://babin.perso.enseeiht.fr/r/thesis/`

```
THEORY
        //   Real Theory
 Real   //   Jean-Raymond Abrial, Michael Butler
        //   June 2014
AXIOMATIC DEFINITIONS
  real_def
    TYPES
      REAL
    OPERATORS
      • plus:  plus(a : REAL, b : REAL) EXPRESSION INFIX  REAL
      • zero:  zero() EXPRESSION PREFIX  REAL
      • minus:  minus(a : REAL) EXPRESSION PREFIX  REAL
      • mult:  mult(a : REAL, b : REAL) EXPRESSION INFIX  REAL
      • one:  one() EXPRESSION PREFIX  REAL
      • inv:  inv(a : REAL) EXPRESSION PREFIX  REAL
              well-definedness condition
                a ≠ zero
      • leq:  leq(a : REAL, b : REAL) PREDICATE PREFIX
      • sup:  sup(A : ℙ(REAL)) EXPRESSION PREFIX  REAL
              well-definedness condition
                A≠∅   //   A is not empty
                ∃m · m∈REAL ∧ (∀x · x∈A ⇒ leq(x,m))   //   A has an upper bound
      • inf:  inf(A : ℙ(REAL)) EXPRESSION PREFIX  REAL
              well-definedness condition
                A≠∅   //   A is not empty
                ∃m · m∈REAL ∧ (∀x · x∈A ⇒ leq(m,x))   //   A has a lower bound
      • smr:  smr(a : REAL, b : REAL) PREDICATE PREFIX
      • sub:  sub(a : REAL, b : REAL) EXPRESSION INFIX  REAL
      • cnt:  cnt(f : ℙ(REAL×REAL), x : REAL) PREDICATE PREFIX
              well-definedness condition
                f ∈ REAL⟶REAL
      • gtr:  gtr(a : REAL, b : REAL) PREDICATE PREFIX
    AXIOMS
```

axm1:	∀x,y · (x plus y) = (y plus x)	*addition is commutative*
axm2:	∀x,y,z · ((x plus y) plus z) = (x plus (y plus z))	*addition is associative*
axm3:	∀x · (x plus zero) = x	*addition has an identity*
axm4:	∀x · (x plus (minus (x))) = zero	*addition has an inverse*
axm5:	∀x,y · (x mult y) = (y mult x)	*multiplication is commutative*
axm6:	∀x,y,z · ((x mult y) mult z) = (x mult (y mult z))	*multiplication is associative*
axm7:	∀x · (x mult one) = x	*multiplication has an identity*
axm8:	∀x · x≠zero ⇒ (x mult (inv (x))) = one	*multiplication has an inverse (except for zero)*
axm9:	zero ≠ one	*zero different from one*
axm10:	∀x,y,z · (x mult (y plus z)) = ((x mult y) plus (x mult z))	*multiplication is distributive over addition*
axm11:	∀x · leq(x,x)	*order is reflexive*
axm12:	∀x,y · leq(x,y) ∧ leq(y,x) ⇒ x=y	*order is antisymmetric*
axm13:	∀x,y,z · leq(x,y) ∧ leq(y,z) ⇒ leq(x,z)	*order is transitive*
axm14:	∀x,y · leq(x,y) ∨ leq(y,x)	*order is total*
axm15:	∀x,y,z · leq(x,y) ⇒ leq(x plus z, y plus z)	*order is compatible with addition*
axm16:	∀x,y,z · leq(x,y) ∧ leq(zero,z) ∧ z≠zero ⇒ leq(x mult z, y mult z)	*order is compatible with positive multiplication*

```
        ∀A · A⊆REAL ∧
             A≠∅ ∧
axm17:       (∃m · m∈REAL ∧ (∀x · x∈A ⇒ leq(x,m)))          sup(A) is an upper bound of A
           ⇒
             (∀x · x∈A ⇒ leq(x,sup(A)))
        ∀A,v · A⊆REAL ∧
               A≠∅ ∧
axm18:         (∃m · m∈REAL ∧ (∀x · x∈A ⇒ leq(x,m))) ∧      sup(A) is the least upper
               (∀x · x∈A ⇒ leq(x,v))                        bound of A
             ⇒
               leq(sup(A),v)
        ∀A · A⊆REAL ∧
             A≠∅ ∧
axm19:       (∃m · m∈REAL ∧ (∀x · x∈A ⇒ leq(m,x)))          inf(A) is a lower bound of A
           ⇒
             (∀x · x∈A ⇒ leq(inf(A),x))
        ∀A,v · A⊆REAL ∧
               A≠∅ ∧
axm20:         (∃m · m∈REAL ∧ (∀x · x∈A ⇒ leq(m,x))) ∧      inf(A) is the greatest
               (∀x · x∈A ⇒ leq(v,x))                        lower bound of A
             ⇒
               leq(v,inf(A))
axm21:  ∀x,y · smr(x,y) ⇔ leq(x,y) ∧ x≠y                    Definition of relation
                                                            "strictly smaller"
axm24:  ∀x,y · gtr(x,y) ⇔ leq(y,x) ∧ x≠y                    Definition of relation
                                                            "strictly greater"
axm22:  ∀x,y · (x sub y) = (x plus minus(y))                Definition of subtraction
        ∀f,c · f∈REAL→REAL ∧ c∈REAL ∧ cnt(f,c)
             ⇒
               (∀e · smr(zero,e)
                     ⇒
                       (∃d · smr(zero,d) ∧
                             (∀x · smr(c sub d,x) ∧
axm23:                             smr(x,c plus d)          Definition of continuity
                                 ⇒
                                   smr(f(c) sub e,f(x)) ∧
                                   smr(f(x),f(c) plus e)
                             )
                       )
               )
PROOF RULES
   add_com :
    Metavariables
     ▪ a ∈ REAL
     ▪ b ∈ REAL
    Rewrite Rules
     • rew1 : a plus b      (case-incomplete, interactive)      add_com
       ▪ rhs1   :   ⊤   ►   b plus a
```

```
add_assoc :
 Metavariables
  ▪ x ∈ REAL
  ▪ y ∈ REAL
  ▪ z ∈ REAL
 Rewrite Rules
  • rew2 : (x plus y) plus z      (case-incomplete, interactive)      add_assoc
   ▪ rhs1   :   ⊤   ►   x plus (y plus z)
add_id :
 Metavariables
  ▪ x ∈ REAL
 Rewrite Rules
  • rew3 : x plus zero      (case-incomplete, interactive)      add_id
   ▪ rhs1   :   ⊤   ►   x
add_inv :
 Metavariables
  ▪ x ∈ REAL
 Rewrite Rules
  • rew4 : x plus minus(x)      (case-incomplete, interactive)      add_inv
   ▪ rhs1   :   ⊤   ►   zero
add_assoc2 :
 Metavariables
  ▪ x ∈ REAL
  ▪ y ∈ REAL
  ▪ z ∈ REAL
 Rewrite Rules
  • rew5 : x plus (y plus z)      (case-incomplete, interactive)      add_assoc2
   ▪ rhs1   :   ⊤   ►   (x plus y) plus z
add_id2 :
 Metavariables
  ▪ x ∈ REAL
 Rewrite Rules
  • rew6 : zero plus x      (case-incomplete, interactive)      add_id2
   ▪ rhs1   :   ⊤   ►   x
add_inv2 :
 Metavariables
  ▪ x ∈ REAL
 Rewrite Rules
  • rew7 : minus(x) plus x      (case-incomplete, interactive)      add_inv2
   ▪ rhs1   :   ⊤   ►   zero
mult_com :
 Metavariables
  ▪ x ∈ REAL
  ▪ y ∈ REAL
 Rewrite Rules
  • rew8 : x mult y      (case-incomplete, interactive)      mult_com
   ▪ rhs1   :   ⊤   ►   y mult x
```

```
mult_assoc :
 Metavariables
  ▪ x ∊ REAL
  ▪ y ∊ REAL
  ▪ z ∊ REAL
 Rewrite Rules
  • rew9 : (x mult y) mult z      (case-incomplete, interactive)      mult_assoc
    ▪ rhs1   :   ⊤   ►   x mult (y mult z)
mult_id :
 Metavariables
  ▪ x ∊ REAL
 Rewrite Rules
  • rew10 : x mult one      (case-incomplete, interactive)      mult_id
    ▪ rhs1   :   ⊤    ►   x
mult_inv :
 Metavariables
  ▪ x ∊ REAL
 Rewrite Rules
  • rew11 : x mult inv(x)      (case-incomplete, interactive)      mult_inv
    ▪ rhs1   :   x≠zero   ►   one
mult_assoc2 :
 Metavariables
  ▪ x ∊ REAL
  ▪ y ∊ REAL
  ▪ z ∊ REAL
 Rewrite Rules
  • rew12 : x mult (y mult z)      (case-incomplete, interactive)      mult_assoc2
    ▪ rhs1   :   ⊤   ►   (x mult y) mult z
mult_id2 :
 Metavariables
  ▪ x ∊ REAL
 Rewrite Rules
  • rew13 : one mult x      (case-incomplete, interactive)      mult_id2
    ▪ rhs1   :   ⊤   ►   x
mult_inv2 :
 Metavariables
  ▪ x ∊ REAL
 Rewrite Rules
  • rew14 : inv(x) mult x      (case-incomplete, interactive)      mult_inv2
    ▪ rhs1   :   x≠zero   ►   one
mult_distrib :
 Metavariables
  ▪ x ∊ REAL
  ▪ y ∊ REAL
  ▪ z ∊ REAL
 Rewrite Rules
  • rew15 : x mult (y plus z)      (case-incomplete, interactive)      mult_distrib
    ▪ rhs1   :   ⊤   ►   (x mult y) plus (x mult z)
```

```
mult_distrib2 :
 Metavariables
  ▪ x ∈ REAL
  ▪ y ∈ REAL
  ▪ z ∈ REAL
 Rewrite Rules
  • rew16 : (x plus y) mult z      (case-incomplete, interactive)      mult_distrib2
   ▪ rhs1   :   ⊤   ►   (x mult z) plus (y mult z)
sub_plus :
 Metavariables
  ▪ x ∈ REAL
  ▪ y ∈ REAL
 Rewrite Rules
  • rew19 : x sub y      (case-incomplete, interactive)      sub_plus
   ▪ rhs1   :   ⊤   ►   x plus minus(y)
gtr_smr :
 Metavariables
  ▪ x ∈ REAL
  ▪ y ∈ REAL
 Rewrite Rules
  • rew20 : gtr(x,y)      (case-incomplete, interactive)      gtr_smr
   ▪ rhs1   :   ⊤   ►   smr(y,x)
END
```

```
THEORY
 RealPos
IMPORTS THEORY PROJECTS
  [RealTheory]
   THEORIES
    Real
AXIOMATIC DEFINITIONS
  real_pos_def
   OPERATORS
    • cnt_int:  cnt_int(f : ℙ(REAL×REAL), a : REAL, b : REAL) PREDICATE PREFIX
          well-definedness condition
           f ∈ REAL ⇸ REAL
           a ∈ REAL
           b ∈ REAL
           leq(a,b)
           {x | x ∈ REAL ∧ leq(a,x) ∧ leq(x,b)} ⊆ dom(f)
   AXIOMS
    axm1: ∀f,a,b · f ∈ REAL ⇸ REAL                                   //   Definition of
                 ∧ a ∈ REAL                                            //   continuity
                 ∧ b ∈ REAL                                            //   on an interval
                 ∧ leq(a,b)
                 ∧ {x | x ∈ REAL ∧ leq(a,x) ∧ leq(x,b)} ⊆ dom(f) ⇒
          (cnt_int(f,a,b)
             ⇔
             (∀c · leq(a,c) ∧ leq(c,b) ⇒
                (∀e· smr(zero,e)
                   ⇒
                   (∃d·smr(zero,d) ∧
                      (∀x· leq(a,x) ∧ leq(x,b) ⇒
                         (smr(c sub d,x) ∧
                          smr(x,c plus d)
                          ⇒
                          smr(f(c) sub e,f(x)) ∧
                          smr(f(x),f(c) plus e))
                      )
                   )
                )
             )
          )
END
```

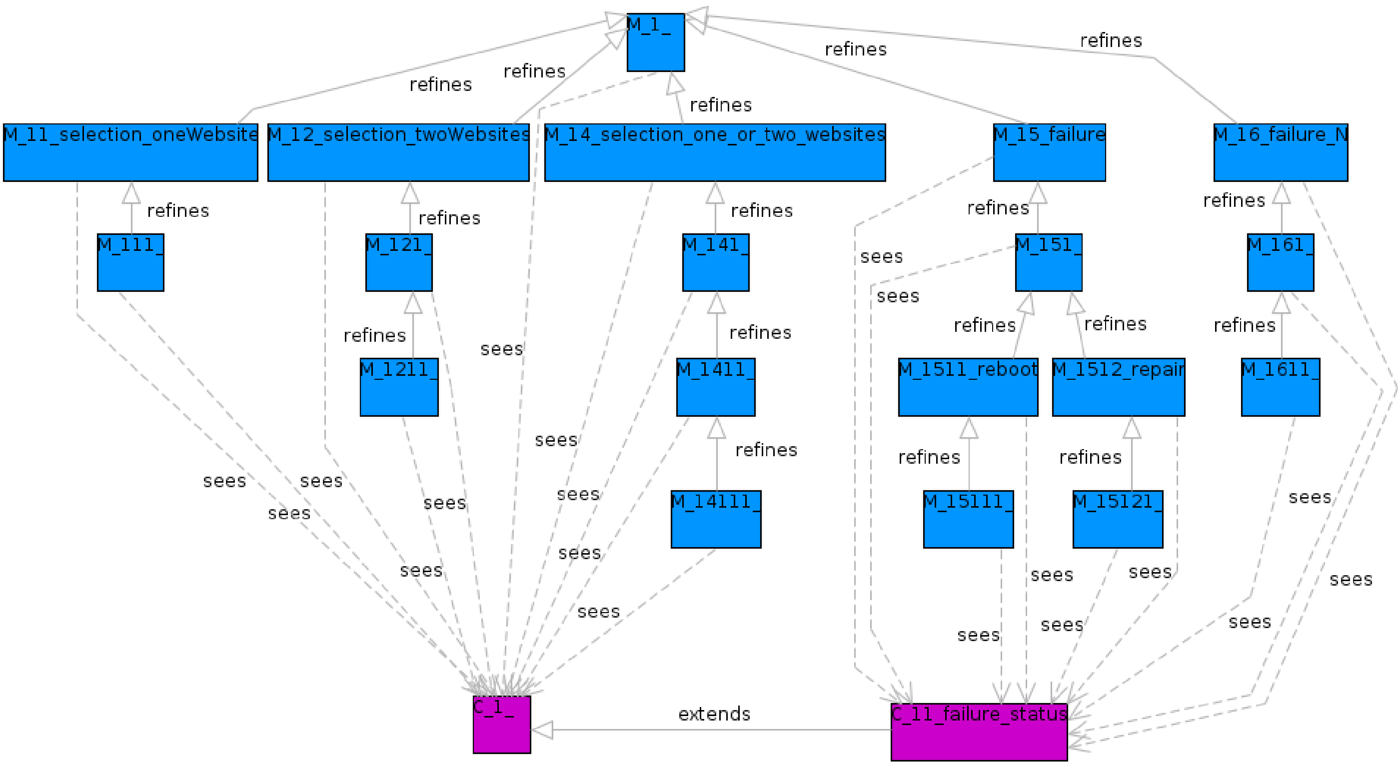
B Discrete systems substitution

In Chapter 5, a simplified version of the tree of machines is presented:

- *M0* corresponds to *M_1* in the complete models.
- *R1* corresponds to *M_11* & *M_111* combined.
- *R2* corresponds to *M_12*, *M_121* & *M_1211* combined.
- *R3* corresponds to *M_15*, *M_151*, *M_1512* & *M_15121* combined.

Components:

- WS1 and WS2, with failures
  - *M_15_failure* (page 178)
  - *M_151_* (page 181)
  - Using reboot
    - *M_1511_reboot* (page 185)
    - *M_15111_* (page 190)
  - Using repair
    - *M_1512_repair* (page 196)
    - *M_15121_* (page 202)
- N systems, with failures
  - *M_16_failure_N* (page 211)
  - *M_161_* (page 213)
  - *M_1611_* (page 215)



The models are also available at: http://babin.perso.enseeiht.fr/r/thesis/

CONTEXT C_1_

SETS

PRODUCTS all the products in the world

SITES all the sites in the world

CONSTANTS

STOCKS

P products we want to buy

AXIOMS

axm1: $\mathrm{finite}(PRODUCTS)$

axm2: $\mathrm{finite}(SITES)$

axm3: $\mathrm{card}(SITES) \geq 2$

axm4: $STOCKS = SITES \times PRODUCTS$

axm5: $P \subseteq PRODUCTS$

END

CONTEXT C_11_failure_status
EXTENDS C_1_
SETS
FAILURE_STATUS
CONSTANTS
OK
NOT_OK
AXIOMS
axm1: partition($FAILURE\_STATUS, \{OK\}, \{NOT\_OK\}$)
END

MACHINE M_1_
SEES C_1_
VARIABLES

var_M_1_seq -

carts -

INVARIANTS

type1: $var\_M\_1\_seq \in \mathbb{N}$

type2: ⟨theorem⟩ $P \subseteq PRODUCTS$

type3: $carts \subseteq STOCKS$

prop1: $(var\_M\_1\_seq < 4) \Rightarrow \mathrm{ran}(carts) = P$

we have all the products we wanted in our carts after the 'selection' step

prop2: $\forall p \cdot p \in \mathrm{ran}(carts) \Rightarrow \mathrm{card}(carts^{-1}[\{p\}]) = 1$

each product has been selected in only one site

DLF_1: $\neg(\exists someCarts \cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ someCarts \subseteq SITES \times P$
$\wedge\ \mathrm{ran}(someCarts) = P$
$\wedge\ (\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1))$
$\vee\ var\_M\_1\_seq = 3$
$\vee\ var\_M\_1\_seq = 2$
$\vee\ var\_M\_1\_seq = 1)$
$\Rightarrow$
$var\_M\_1\_seq = 0$

(deadlock => finished)

VARIANT

$var\_M\_1\_seq$

EVENTS

Initialisation

begin

act1: $var\_M\_1\_seq := 4$

act3: $carts := \varnothing$

end

Event selection ⟨convergent⟩ $\widehat{=}$

any

someCarts

where

grd1: $var\_M\_1\_seq = 4$

grd2: $someCarts \subseteq SITES \times P$

grd3: $\mathrm{ran}(someCarts) = P$

grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

act2: $carts := someCarts$

end

Event payment ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 3$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event billing ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 2$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event delivery ⟨convergent⟩ $\widehat{=}$
when
grd1: $var\_M\_1\_seq = 1$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
END

MACHINE M_11_selection_oneWebsite
REFINES M_1_
SEES C_1_
VARIABLES

var_M_1_seq -

carts -

carts_ref -

var_M_11_loop -

site -

INVARIANTS

type1: $carts\_ref \subseteq SITES \times P$

type2: $var\_M\_11\_loop \in \mathbb{N}$

type3: $site \in SITES$

VARIANT

$var\_M\_1\_seq + var\_M\_11\_loop$

EVENTS

Initialisation

begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_11\_loop := \mathrm{card}(P)$
act3: $carts := \varnothing$
act4: $carts\_ref := \varnothing$
act5: $site :\in SITES$

end

Event addItemToCart_loop ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_11\_loop > 0$
grd3: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_11\_loop := var\_M\_11\_loop - 1$
act2: $carts\_ref := carts\_ref \cup \{site \mapsto someProduct\}$

end

Event confirmCarts ⟨convergent⟩ $\widehat{=}$
refines selection

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_11\_loop = 0$
grd3: $\mathrm{ran}(carts\_ref) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow carts\_ref^{-1}[\{p\}] = \{site\}$

with

someCarts: $someCarts = carts\_ref$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := carts\_ref$

end

Event payment ⟨convergent⟩ $\widehat{=}$
extends payment

when

grd1: $var\_M\_1\_seq = 3$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
when
grd1: $var\_M\_1\_seq = 2$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery
when
grd1: $var\_M\_1\_seq = 1$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
END

MACHINE M_111_
REFINES M_11_selection_oneWebsite
SEES C_1_
VARIABLES

var_M_1_seq -

carts -

carts_ref -

var_M_11_loop -

site -

var_M_111_seq -

selectedItem -

INVARIANTS

type1: $var\_M\_111\_seq \in \mathbb{N}$

type2: $selectedItem \in \mathbb{P}(P)$

prop1: $var\_M\_111\_seq \geq 1 \Rightarrow \mathrm{card}(selectedItem) = 0$

prop2: $var\_M\_111\_seq < 1 \Rightarrow \mathrm{card}(selectedItem) = 1$

VARIANT

$var\_M\_1\_seq + var\_M\_11\_loop + var\_M\_111\_seq$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$

act2: $var\_M\_11\_loop := \mathrm{card}(P)$

act3: $carts := \varnothing$

act4: $carts\_ref := \varnothing$

act5: $site :\in SITES$

act6: $var\_M\_111\_seq := 1$

act7: $selectedItem := \varnothing$

end

Event selectItemInItemList ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_11\_loop > 0$

grd3: $var\_M\_111\_seq = 1$

grd4: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_111\_seq := var\_M\_111\_seq - 1$

act2: $selectedItem := \{someProduct\}$

end

Event addSelectedItemToCart ⟨convergent⟩ $\widehat{=}$

refines addItemToCart_loop

any

item used to access the element in selectedItem

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_11\_loop > 0$

grd3: $var\_M\_111\_seq = 0$

grd4: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \land selectedItem = \{p\}$

grd5: $selectedItem = \{item\}$

with
someProduct: $selectedItem = \{someProduct\}$
then
act1: $var\_M\_11\_loop := var\_M\_11\_loop - 1$
act2: $carts\_ref := carts\_ref \cup \{site \mapsto item\}$
end
Event selection ⟨convergent⟩ $\widehat{=}$
extends confirmCarts
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_11\_loop = 0$
grd3: $\mathrm{ran}(carts\_ref) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow carts\_ref^{-1}[\{p\}] = \{site\}$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := carts\_ref$
end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
when
grd1: $var\_M\_1\_seq = 3$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
when
grd1: $var\_M\_1\_seq = 2$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery
when
grd1: $var\_M\_1\_seq = 1$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
END

MACHINE M_12_selection_twoWebsites
REFINES M_1_
SEES C_1_
VARIABLES

var_M_1_seq -

carts -

var_M_12_par_A -

var_M_12_par_B -

INVARIANTS

type1: $var\_M\_12\_par\_A \in \mathbb{N}$

type2: $var\_M\_12\_par\_B \in \mathbb{N}$

VARIANT

$var\_M\_1\_seq + var\_M\_12\_par\_A + var\_M\_12\_par\_B$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$

act3: $carts := \varnothing$

act4: $var\_M\_12\_par\_A := 1$

act5: $var\_M\_12\_par\_B := 1$

end

Event selection_A ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_12\_par\_A = 1$

then

act1: $var\_M\_12\_par\_A := var\_M\_12\_par\_A - 1$

end

Event selection_B ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_12\_par\_B = 1$

then

act1: $var\_M\_12\_par\_B := var\_M\_12\_par\_B - 1$

end

Event selection_join_A_B ⟨convergent⟩ $\widehat{=}$
refines selection

any

someCarts

where

grd1: $var\_M\_1\_seq = 4$

grd2: $someCarts \subseteq SITES \times P$

grd3: $\mathrm{ran}(someCarts) = P$

grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$

grd5: $var\_M\_12\_par\_A = 0$

grd6: $var\_M\_12\_par\_B = 0$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

act2: $carts := someCarts$

end

Event payment ⟨convergent⟩ $\widehat{=}$

extends payment
    when
        grd1: $var\_M\_1\_seq = 3$
    then
        act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
    end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
    when
        grd1: $var\_M\_1\_seq = 2$
    then
        act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
    end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery
    when
        grd1: $var\_M\_1\_seq = 1$
    then
        act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
    end
END

MACHINE M_121_
REFINES M_12_selection_twoWebsites
SEES C_1_
VARIABLES

var_M_1_seq -
carts -
carts_ref -
var_M_12_par_A -
var_M_12_par_B -
site_A -
site_B -
var_M_121_loop_A -
var_M_121_loop_B -

INVARIANTS

type1: $carts\_ref \subseteq SITES \times P$
type2: $var\_M\_121\_loop\_A \in \mathbb{N}$
type3: $var\_M\_121\_loop\_B \in \mathbb{N}$
type4: $site\_A \in SITES$
type5: $site\_B \in SITES$

VARIANT

$var\_M\_1\_seq+var\_M\_12\_par\_A+var\_M\_12\_par\_B+var\_M\_121\_loop\_A+var\_M\_121\_loop\_B$

EVENTS
Initialisation
begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_121\_loop\_A, var\_M\_121\_loop\_B :\mid$
$var\_M\_121\_loop\_A' + var\_M\_121\_loop\_B' = \mathrm{card}(P)$
$\wedge\, var\_M\_121\_loop\_A' \in \mathbb{N}$
$\wedge\, var\_M\_121\_loop\_B' \in \mathbb{N}$
act3: $carts := \varnothing$
act4: $var\_M\_12\_par\_A := 1$
act5: $var\_M\_12\_par\_B := 1$
act6: $carts\_ref := \varnothing$
act7: $site\_A :\in SITES$
act8: $site\_B :\in SITES$

end
Event selection_A_loop ⟨convergent⟩ $\widehat{=}$
any

someProduct

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_12\_par\_A = 1$
grd3: $var\_M\_121\_loop\_A > 0$
grd4: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_121\_loop\_A := var\_M\_121\_loop\_A - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_A \mapsto someProduct\}$

end
Event selection_A_loop_end ⟨convergent⟩ $\widehat{=}$
extends selection_A

when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_12\_par\_A = 1$
grd3: $var\_M\_121\_loop\_A = 0$
then
act1: $var\_M\_12\_par\_A := var\_M\_12\_par\_A - 1$
end
Event selection_B_loop ⟨convergent⟩ $\widehat{=}$
any
someProduct
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_12\_par\_B = 1$
grd3: $var\_M\_121\_loop\_B > 0$
grd4: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$
then
act1: $var\_M\_121\_loop\_B := var\_M\_121\_loop\_B - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_B \mapsto someProduct\}$
end
Event selection_B_loop_end ⟨convergent⟩ $\widehat{=}$
extends selection_B
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_12\_par\_B = 1$
grd3: $var\_M\_121\_loop\_B = 0$
then
act1: $var\_M\_12\_par\_B := var\_M\_12\_par\_B - 1$
end
Event confirmCarts ⟨convergent⟩ $\widehat{=}$
refines selection_join_A_B
when
grd1: $var\_M\_1\_seq = 4$
grd2: $\mathrm{ran}(carts\_ref) = P$
grd3: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow$
$(carts\_ref^{-1}[\{p\}] = \{site\_A\} \lor carts\_ref^{-1}[\{p\}] = \{site\_B\})$
grd4: $var\_M\_12\_par\_A = 0$
grd5: $var\_M\_12\_par\_B = 0$
grd6: $var\_M\_121\_loop\_A = 0$
grd7: $var\_M\_121\_loop\_B = 0$
with
someCarts: $someCarts = carts\_ref$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := carts\_ref$
end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
when
grd1: $var\_M\_1\_seq = 3$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event billing ⟨convergent⟩ $\widehat{=}$

extends billing

  when

    grd1: $var\_M\_1\_seq = 2$

  then

    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

  end

Event delivery ⟨convergent⟩ $\widehat{=}$

extends delivery

  when

    grd1: $var\_M\_1\_seq = 1$

  then

    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

  end

END

MACHINE M_1211_
REFINES M_121_
SEES C_1_
VARIABLES

var_M_1_seq -
carts -
carts_ref -
var_M_12_par_A -
var_M_12_par_B -
site_A -
site_B -
var_M_121_loop_A -
var_M_121_loop_B -
var_M_1211_seq_A -
var_M_1211_seq_B -
selectedItem_A -
selectedItem_B -

INVARIANTS

type1: $var\_M\_1211\_seq\_A \in \mathbb{N}$
type2: $var\_M\_1211\_seq\_B \in \mathbb{N}$
type3: $selectedItem\_A \in \mathbb{P}(P)$
type4: $selectedItem\_B \in \mathbb{P}(P)$
prop1: $var\_M\_1211\_seq\_A \geq 1 \Rightarrow \mathrm{card}(selectedItem\_A) = 0$
prop2: $var\_M\_1211\_seq\_A < 1 \Rightarrow \mathrm{card}(selectedItem\_A) = 1$
prop3: $var\_M\_1211\_seq\_B \geq 1 \Rightarrow \mathrm{card}(selectedItem\_B) = 0$
prop4: $var\_M\_1211\_seq\_B < 1 \Rightarrow \mathrm{card}(selectedItem\_B) = 1$

VARIANT

$var\_M\_1\_seq + var\_M\_12\_par\_A + var\_M\_12\_par\_B + var\_M\_121\_loop\_A + var\_M\_121\_loop\_B + var\_M\_1211\_seq\_A + var\_M\_1211\_seq\_B$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_121\_loop\_A, var\_M\_121\_loop\_B :| var\_M\_121\_loop\_A' + var\_M\_121\_loop\_B' = \mathrm{card}(P) \land var\_M\_121\_loop\_A' \in \mathbb{N} \land var\_M\_121\_loop\_B' \in \mathbb{N}$
act3: $carts := \varnothing$
act4: $var\_M\_12\_par\_A := 1$
act5: $var\_M\_12\_par\_B := 1$
act6: $carts\_ref := \varnothing$
act7: $site\_A :\in SITES$
act8: $site\_B :\in SITES$
act9: $var\_M\_1211\_seq\_A := 1$
act10: $selectedItem\_A := \varnothing$
act11: $var\_M\_1211\_seq\_B := 1$
act12: $selectedItem\_B := \varnothing$

end

Event selectItemInItemList_A ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_12\_par\_A = 1$

grd3: $var\_M\_121\_loop\_A > 0$

grd4: $var\_M\_1211\_seq\_A = 1$

grd5: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_1211\_seq\_A := var\_M\_1211\_seq\_A - 1$

act2: $selectedItem\_A := \{someProduct\}$

end

Event addSelectedItemToCart_A ⟨convergent⟩ $\widehat{=}$

refines selection_A_loop

any

item used to access the element in selectedItem_A

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_12\_par\_A = 1$

grd3: $var\_M\_121\_loop\_A > 0$

grd4: $var\_M\_1211\_seq\_A = 0$

grd5: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \land selectedItem\_A = \{p\}$

grd6: $selectedItem\_A = \{item\}$

with

someProduct: $selectedItem\_A = \{someProduct\}$

then

act1: $var\_M\_121\_loop\_A := var\_M\_121\_loop\_A - 1$

act2: $carts\_ref := carts\_ref \cup \{site\_A \mapsto item\}$

end

Event selection_A_loop_end ⟨convergent⟩ $\widehat{=}$

extends selection_A_loop_end

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_12\_par\_A = 1$

grd3: $var\_M\_121\_loop\_A = 0$

then

act1: $var\_M\_12\_par\_A := var\_M\_12\_par\_A - 1$

end

Event selectItemInItemList_B ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_12\_par\_B = 1$

grd3: $var\_M\_121\_loop\_B > 0$

grd4: $var\_M\_1211\_seq\_B = 1$

grd5: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_1211\_seq\_B := var\_M\_1211\_seq\_B - 1$

act2: $selectedItem\_B := \{someProduct\}$

end

Event addSelectedItemToCart_B ⟨convergent⟩ $\widehat{=}$

refines selection_B_loop

any

item used to access the element in selectedItem_B

where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_12\_par\_B = 1$
grd3: $var\_M\_121\_loop\_B > 0$
grd4: $var\_M\_1211\_seq\_B = 0$
grd5: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_B = \{p\}$
grd6: $selectedItem\_B = \{item\}$
with
someProduct: $selectedItem\_B = \{someProduct\}$
then
act1: $var\_M\_121\_loop\_B := var\_M\_121\_loop\_B - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_B \mapsto item\}$
end
Event selection_B_loop_end ⟨convergent⟩ $\widehat{=}$
extends selection_B_loop_end
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_12\_par\_B = 1$
grd3: $var\_M\_121\_loop\_B = 0$
then
act1: $var\_M\_12\_par\_B := var\_M\_12\_par\_B - 1$
end
Event confirmCarts ⟨convergent⟩ $\widehat{=}$
extends confirmCarts
when
grd1: $var\_M\_1\_seq = 4$
grd2: $\mathrm{ran}(carts\_ref) = P$
grd3: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow$
$(carts\_ref^{-1}[\{p\}] = \{site\_A\} \vee carts\_ref^{-1}[\{p\}] = \{site\_B\})$
grd4: $var\_M\_12\_par\_A = 0$
grd5: $var\_M\_12\_par\_B = 0$
grd6: $var\_M\_121\_loop\_A = 0$
grd7: $var\_M\_121\_loop\_B = 0$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := carts\_ref$
end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
when
grd1: $var\_M\_1\_seq = 3$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
when
grd1: $var\_M\_1\_seq = 2$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery

when

`grd1`: $var\_M\_1\_seq = 1$

then

`act1`: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

END

MACHINE M_14_selection_one_or_two_websites
REFINES M_1_
SEES C_1_
VARIABLES
var_M_1_seq -
carts -
var_M_14_cho -
INVARIANTS
type1: $var\_M\_14\_cho \in \mathbb{N}$
VARIANT
$var\_M\_1\_seq + var\_M\_14\_cho$
EVENTS
Initialisation ⟨extended⟩
begin
act1: $var\_M\_1\_seq := 4$
act3: $carts := \varnothing$
act4: $var\_M\_14\_cho :\in \{1, 2\}$
end
Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_14\_cho = 1$
then
act1: $var\_M\_14\_cho := 0$
end
Event selection_twoWebsites ⟨convergent⟩ $\widehat{=}$
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_14\_cho = 2$
then
act1: $var\_M\_14\_cho := 0$
end
Event selection ⟨convergent⟩ $\widehat{=}$
extends selection
any
$someCarts$
where
grd1: $var\_M\_1\_seq = 4$
grd2: $someCarts \subseteq SITES \times P$
grd3: $\mathrm{ran}(someCarts) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$
grd5: $var\_M\_14\_cho = 0$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := someCarts$
end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
when
grd1: $var\_M\_1\_seq = 3$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event billing ⟨convergent⟩ $\widehat{=}$

extends billing

when

`grd1`: $var\_M\_1\_seq = 2$

then

`act1`: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event delivery ⟨convergent⟩ $\widehat{=}$

extends delivery

when

`grd1`: $var\_M\_1\_seq = 1$

then

`act1`: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

END

MACHINE M_141_
REFINES M_14_selection_one_or_two_websites
SEES C_1_
VARIABLES
    var_M_1_seq -
    carts -
    var_M_14_cho -
    var_M_141_par_A -
    var_M_141_par_B -
INVARIANTS
    type1: $var\_M\_141\_par\_A \in \mathbb{N}$
    type2: $var\_M\_141\_par\_B \in \mathbb{N}$
VARIANT
    $var\_M\_1\_seq + var\_M\_14\_cho + var\_M\_141\_par\_A + var\_M\_141\_par\_B$
EVENTS
Initialisation ⟨extended⟩
  begin
    act1: $var\_M\_1\_seq := 4$
    act3: $carts := \varnothing$
    act4: $var\_M\_14\_cho :\in \{1, 2\}$
    act5: $var\_M\_141\_par\_A := 1$
    act6: $var\_M\_141\_par\_B := 1$
  end
Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$
extends selection_oneWebsite
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_14\_cho = 1$
  then
    act1: $var\_M\_14\_cho := 0$
  end
Event selection_twoWebsites_A ⟨convergent⟩ $\widehat{=}$
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_14\_cho = 2$
    grd3: $var\_M\_141\_par\_A = 1$
  then
    act1: $var\_M\_141\_par\_A := var\_M\_141\_par\_A - 1$
  end
Event selection_twoWebsites_B ⟨convergent⟩ $\widehat{=}$
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_14\_cho = 2$
    grd3: $var\_M\_141\_par\_B = 1$
  then
    act1: $var\_M\_141\_par\_B := var\_M\_141\_par\_B - 1$
  end
Event selection_twoWebsites_join_A_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites
  when
    grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$
grd3: $var\_M\_141\_par\_A = 0$
grd4: $var\_M\_141\_par\_B = 0$

then

act1: $var\_M\_14\_cho := 0$

end

Event selection ⟨convergent⟩ $\widehat{=}$

extends selection

any

$someCarts$

where

grd1: $var\_M\_1\_seq = 4$
grd2: $someCarts \subseteq SITES \times P$
grd3: $\mathrm{ran}(someCarts) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$
grd5: $var\_M\_14\_cho = 0$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := someCarts$

end

Event payment ⟨convergent⟩ $\widehat{=}$

extends payment

when

grd1: $var\_M\_1\_seq = 3$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event billing ⟨convergent⟩ $\widehat{=}$

extends billing

when

grd1: $var\_M\_1\_seq = 2$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event delivery ⟨convergent⟩ $\widehat{=}$

extends delivery

when

grd1: $var\_M\_1\_seq = 1$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

END

MACHINE M_1411_
REFINES M_141_
SEES C_1_
VARIABLES

var_M_1_seq -
carts -
var_M_14_cho -
var_M_141_par_A -
var_M_141_par_B -
carts_ref -
var_M_1411_loop_1 -
var_M_1411_loop_2_A -
var_M_1411_loop_2_B -
site_1 -
site_2_A -
site_2_B -

INVARIANTS

type1: $carts\_ref \subseteq SITES \times P$
type2: $var\_M\_1411\_loop\_1 \in \mathbb{N}$
type3: $var\_M\_1411\_loop\_2\_A \in \mathbb{N}$
type4: $var\_M\_1411\_loop\_2\_B \in \mathbb{N}$
type5: $site\_1 \in SITES$
type6: $site\_2\_A \in SITES$
type7: $site\_2\_B \in SITES$
prop1: $var\_M\_14\_cho = 1 \Rightarrow \mathrm{dom}(carts\_ref) \subseteq \{site\_1\}$
prop2: $var\_M\_14\_cho = 2 \Rightarrow \mathrm{dom}(carts\_ref) \subseteq \{site\_2\_A, site\_2\_B\}$

VARIANT

$var\_M\_1\_seq + var\_M\_14\_cho + var\_M\_141\_par\_A + var\_M\_141\_par\_B + var\_M\_1411\_loop\_1 + var\_M\_1411\_loop\_2\_A + var\_M\_1411\_loop\_2\_B$

EVENTS

Initialisation

begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_1411\_loop\_1, var\_M\_1411\_loop\_2\_A, var\_M\_1411\_loop\_2\_B :\mid$
$var\_M\_1411\_loop\_1' = \mathrm{card}(P)$
$\land\ var\_M\_1411\_loop\_2\_A' + var\_M\_1411\_loop\_2\_B' = \mathrm{card}(P)$
$\land\ var\_M\_1411\_loop\_2\_A' \in \mathbb{N}$
$\land\ var\_M\_1411\_loop\_2\_B' \in \mathbb{N}$
act3: $carts := \varnothing$
act4: $var\_M\_14\_cho :\in \{1, 2\}$
act5: $var\_M\_141\_par\_A := 1$
act6: $var\_M\_141\_par\_B := 1$
act7: $carts\_ref := \varnothing$
act8: $site\_1 :\in SITES$
act9: $site\_2\_A :\in SITES$
act10: $site\_2\_B :\in SITES$

end

Event selection_oneWebsite_loop ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 1$

grd3: $var\_M\_1411\_loop\_1 > 0$

grd4: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_1411\_loop\_1 := var\_M\_1411\_loop\_1 - 1$

act2: $carts\_ref := carts\_ref \cup \{site\_1 \mapsto someProduct\}$

end

Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$

extends selection_oneWebsite

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 1$

grd3: $var\_M\_1411\_loop\_1 = 0$

then

act1: $var\_M\_14\_cho := 0$

end

Event selection_twoWebsites_A_loop ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_A = 1$

grd4: $var\_M\_1411\_loop\_2\_A > 0$

grd5: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_1411\_loop\_2\_A := var\_M\_1411\_loop\_2\_A - 1$

act2: $carts\_ref := carts\_ref \cup \{site\_2\_A \mapsto someProduct\}$

end

Event selection_twoWebsites_A ⟨convergent⟩ $\widehat{=}$

extends selection_twoWebsites_A

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_A = 1$

grd4: $var\_M\_1411\_loop\_2\_A = 0$

then

act1: $var\_M\_141\_par\_A := var\_M\_141\_par\_A - 1$

end

Event selection_twoWebsites_B_loop ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_B = 1$

grd4: $var\_M\_1411\_loop\_2\_B > 0$

grd5: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_1411\_loop\_2\_B := var\_M\_1411\_loop\_2\_B - 1$

act2: $carts\_ref := carts\_ref \cup \{site\_2\_B \mapsto someProduct\}$

end

Event selection_twoWebsites_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_B
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_14\_cho = 2$
    grd3: $var\_M\_141\_par\_B = 1$
    grd4: $var\_M\_1411\_loop\_2\_B = 0$
  then
    act1: $var\_M\_141\_par\_B := var\_M\_141\_par\_B - 1$
  end
Event selection_twoWebsites_join_A_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_join_A_B
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_14\_cho = 2$
    grd3: $var\_M\_141\_par\_A = 0$
    grd4: $var\_M\_141\_par\_B = 0$
  then
    act1: $var\_M\_14\_cho := 0$
  end
Event confirmSelection ⟨convergent⟩ $\widehat{=}$
refines selection
  when
    grd1: $var\_M\_1\_seq = 4$
    grd3: $\mathrm{ran}(carts\_ref) = P$
    grd4: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow \mathrm{card}(carts\_ref^{-1}[\{p\}]) = 1$
    grd5: $var\_M\_14\_cho = 0$
  with
    someCarts: $someCarts = carts\_ref$
  then
    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
    act2: $carts := carts\_ref$
  end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
  when
    grd1: $var\_M\_1\_seq = 3$
  then
    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
  end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
  when
    grd1: $var\_M\_1\_seq = 2$
  then
    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
  end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery
  when
    grd1: $var\_M\_1\_seq = 1$
  then
    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
  end
END

MACHINE M_14111_
REFINES M_1411_
SEES C_1_
VARIABLES

var_M_1_seq -
carts -
var_M_14_cho -
var_M_141_par_A -
var_M_141_par_B -
carts_ref -
var_M_1411_loop_1 -
var_M_1411_loop_2_A -
var_M_1411_loop_2_B -
site_1 -
site_2_A -
site_2_B -
var_M_14111_seq_1 -
var_M_14111_seq_2_A -
var_M_14111_seq_2_B -
selectedItem_1 -
selectedItem_2_A -
selectedItem_2_B -

INVARIANTS

type1: $var\_M\_14111\_seq\_1 \in \mathbb{N}$
type2: $var\_M\_14111\_seq\_2\_A \in \mathbb{N}$
type3: $var\_M\_14111\_seq\_2\_B \in \mathbb{N}$
type4: $selectedItem\_1 \in \mathbb{P}(P)$
type5: $selectedItem\_2\_A \in \mathbb{P}(P)$
type6: $selectedItem\_2\_B \in \mathbb{P}(P)$
prop1: $var\_M\_14111\_seq\_1 \geq 1 \Rightarrow \mathrm{card}(selectedItem\_1) = 0$
prop2: $var\_M\_14111\_seq\_1 < 1 \Rightarrow \mathrm{card}(selectedItem\_1) = 1$
prop3: $var\_M\_14111\_seq\_2\_A \geq 1 \Rightarrow \mathrm{card}(selectedItem\_2\_A) = 0$
prop4: $var\_M\_14111\_seq\_2\_A < 1 \Rightarrow \mathrm{card}(selectedItem\_2\_A) = 1$
prop5: $var\_M\_14111\_seq\_2\_B \geq 1 \Rightarrow \mathrm{card}(selectedItem\_2\_B) = 0$
prop6: $var\_M\_14111\_seq\_2\_B < 1 \Rightarrow \mathrm{card}(selectedItem\_2\_B) = 1$

VARIANT

$var\_M\_1\_seq+var\_M\_14\_cho+var\_M\_141\_par\_A+var\_M\_141\_par\_B+var\_M\_1411\_loop\_1+var\_M\_1411\_loop\_2\_A+var\_M\_1411\_loop\_2\_B+var\_M\_14111\_seq\_1+var\_M\_14111\_seq\_2\_A+var\_M\_14111\_seq\_2\_B$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_1411\_loop\_1, var\_M\_1411\_loop\_2\_A, var\_M\_1411\_loop\_2\_B :|$
$var\_M\_1411\_loop\_1' = \mathrm{card}(P)$
$\land\ var\_M\_1411\_loop\_2\_A' + var\_M\_1411\_loop\_2\_B' = \mathrm{card}(P)$
$\land\ var\_M\_1411\_loop\_2\_A' \in \mathbb{N}$
$\land\ var\_M\_1411\_loop\_2\_B' \in \mathbb{N}$
act3: $carts := \varnothing$

act4: $var\_M\_14\_cho :\in \{1, 2\}$
act5: $var\_M\_141\_par\_A := 1$
act6: $var\_M\_141\_par\_B := 1$
act7: $carts\_ref := \varnothing$
act8: $site\_1 :\in SITES$
act9: $site\_2\_A :\in SITES$
act10: $site\_2\_B :\in SITES$
act11: $var\_M\_14111\_seq\_1 := 1$
act12: $selectedItem\_1 := \varnothing$
act13: $var\_M\_14111\_seq\_2\_A := 1$
act14: $selectedItem\_2\_A := \varnothing$
act15: $var\_M\_14111\_seq\_2\_B := 1$
act16: $selectedItem\_2\_B := \varnothing$

end

Event selectItemInItemList_1 ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_14\_cho = 1$
grd3: $var\_M\_1411\_loop\_1 > 0$
grd4: $var\_M\_14111\_seq\_1 = 1$
grd5: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_14111\_seq\_1 := var\_M\_14111\_seq\_1 - 1$
act2: $selectedItem\_1 := \{someProduct\}$

end

Event addSelectedItemToCart_1 ⟨convergent⟩ $\widehat{=}$

refines selection_oneWebsite_loop

any

item used to access the element in selectedItem_1

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_14\_cho = 1$
grd3: $var\_M\_1411\_loop\_1 > 0$
grd4: $var\_M\_14111\_seq\_1 = 0$
grd5: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \land selectedItem\_1 = \{p\}$
grd6: $selectedItem\_1 = \{item\}$

with

someProduct: $selectedItem\_1 = \{someProduct\}$

then

act1: $var\_M\_1411\_loop\_1 := var\_M\_1411\_loop\_1 - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_1 \mapsto item\}$

end

Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$

extends selection_oneWebsite

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_14\_cho = 1$
grd3: $var\_M\_1411\_loop\_1 = 0$

then

act1: $var\_M\_14\_cho := 0$

end

Event selectItemInItemList_2_A ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_A = 1$

grd4: $var\_M\_1411\_loop\_2\_A > 0$

grd5: $var\_M\_14111\_seq\_2\_A = 1$

grd6: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_14111\_seq\_2\_A := var\_M\_14111\_seq\_2\_A - 1$

act2: $selectedItem\_2\_A := \{someProduct\}$

end

Event addSelectedItemToCart_2_A ⟨convergent⟩ $\widehat{=}$

refines selection_twoWebsites_A_loop

any

item used to access the element in selectedItem_2_A

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_A = 1$

grd4: $var\_M\_1411\_loop\_2\_A > 0$

grd5: $var\_M\_14111\_seq\_2\_A = 0$

grd6: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \land selectedItem\_2\_A = \{p\}$

grd7: $selectedItem\_2\_A = \{item\}$

with

someProduct: $selectedItem\_2\_A = \{someProduct\}$

then

act1: $var\_M\_1411\_loop\_2\_A := var\_M\_1411\_loop\_2\_A - 1$

act2: $carts\_ref := carts\_ref \cup \{site\_2\_A \mapsto item\}$

end

Event selection_twoWebsites_A ⟨convergent⟩ $\widehat{=}$

extends selection_twoWebsites_A

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_A = 1$

grd4: $var\_M\_1411\_loop\_2\_A = 0$

then

act1: $var\_M\_141\_par\_A := var\_M\_141\_par\_A - 1$

end

Event selectItemInItemList_2_B ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_B = 1$

grd4: $var\_M\_1411\_loop\_2\_B > 0$

grd5: $var\_M\_14111\_seq\_2\_B = 1$

grd6: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_14111\_seq\_2\_B := var\_M\_14111\_seq\_2\_B - 1$

act2: $selectedItem\_2\_B := \{someProduct\}$

end

Event addSelectedItemToCart_2_B ⟨convergent⟩ $\widehat{=}$

refines selection_twoWebsites_B_loop

any

item used to access the element in selectedItem_2_B

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_B = 1$

grd4: $var\_M\_1411\_loop\_2\_B > 0$

grd5: $var\_M\_14111\_seq\_2\_B = 0$

grd6: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_2\_B = \{p\}$

grd7: $selectedItem\_2\_B = \{item\}$

with

someProduct: $selectedItem\_2\_B = \{someProduct\}$

then

act1: $var\_M\_1411\_loop\_2\_B := var\_M\_1411\_loop\_2\_B - 1$

act2: $carts\_ref := carts\_ref \cup \{site\_2\_B \mapsto item\}$

end

Event selection_twoWebsites_B ⟨convergent⟩ $\widehat{=}$

extends selection_twoWebsites_B

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_B = 1$

grd4: $var\_M\_1411\_loop\_2\_B = 0$

then

act1: $var\_M\_141\_par\_B := var\_M\_141\_par\_B - 1$

end

Event selection_twoWebsites_join_A_B ⟨convergent⟩ $\widehat{=}$

extends selection_twoWebsites_join_A_B

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_14\_cho = 2$

grd3: $var\_M\_141\_par\_A = 0$

grd4: $var\_M\_141\_par\_B = 0$

then

act1: $var\_M\_14\_cho := 0$

end

Event confirmSelection ⟨convergent⟩ $\widehat{=}$

extends confirmSelection

when

grd1: $var\_M\_1\_seq = 4$

grd3: $\mathrm{ran}(carts\_ref) = P$

grd4: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow \mathrm{card}(carts\_ref^{-1}[\{p\}]) = 1$

grd5: $var\_M\_14\_cho = 0$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

act2: $carts := carts\_ref$

end

Event payment ⟨convergent⟩ $\widehat{=}$

extends payment

when

grd1: $var\_M\_1\_seq = 3$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event billing ⟨convergent⟩ $\widehat{=}$

extends billing

when

grd1: $var\_M\_1\_seq = 2$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event delivery ⟨convergent⟩ $\widehat{=}$

extends delivery

when

grd1: $var\_M\_1\_seq = 1$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

END

MACHINE M_15_failure
REFINES M_1_
SEES C_11_failure_status
VARIABLES

var_M_1_seq -
carts -
var_M_15_cho -
failureStatus_1 (one website)
failureStatus_2 (two websites)

INVARIANTS

type1: $var\_M\_15\_cho \in \mathbb{N}$
type2: $failureStatus\_1 \in FAILURE\_STATUS$
type3: $failureStatus\_2 \in FAILURE\_STATUS$
DLF_2:
$$\begin{aligned}
\neg((&var_M_1_seq = 4 \\
&\wedge var_M_15_cho = 1 \\
&\wedge failureStatus_1 = OK) \\
\vee (&var_M_1_seq = 4 \\
&\wedge var_M_15_cho = 1 \\
&\wedge failureStatus_1 = NOT_OK \\
&\wedge failureStatus_2 = OK) \\
\vee (&var_M_1_seq = 4 \\
&\wedge var_M_15_cho = 2 \\
&\wedge failureStatus_2 = OK) \\
\vee (&var_M_1_seq = 4 \\
&\wedge var_M_15_cho = 2 \\
&\wedge failureStatus_2 = NOT_OK \\
&\wedge failureStatus_1 = OK) \\
\vee (&var_M_1_seq = 4 \\
&\wedge var_M_15_cho = 1 \\
&\wedge failureStatus_1 = OK) \\
\vee (&var_M_1_seq = 4 \\
&\wedge var_M_15_cho = 2 \\
&\wedge failureStatus_2 = OK) \\
\vee (&\exists someCarts \cdot \\
&(var_M_1_seq = 4 \\
&\wedge someCarts \subseteq SITES \times P \\
&\wedge \mathrm{ran}(someCarts) = P \\
&\wedge (\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1) \\
&\wedge var_M_15_cho = 0)) \\
\vee\ &var_M_1_seq = 3 \\
\vee\ &var_M_1_seq = 2 \\
\vee\ &var_M_1_seq = 1) \\
\Rightarrow& \\
(&var_M_1_seq = 0 \\
\vee\ &(failureStatus_1 = NOT_OK \wedge failureStatus_2 = NOT_OK))
\end{aligned}$$
deadlock => (finished or total failure)

VARIANT

$var\_M\_1\_seq + var\_M\_15\_cho$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$

act3: $carts := \varnothing$
act4: $var\_M\_15\_cho :\in \{1, 2\}$
act5: $failureStatus\_1 := OK$
act6: $failureStatus\_2 := OK$

end

Event failure_1 ⟨ordinary⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$

then

act1: $failureStatus\_1 := NOT\_OK$

end

Event treat_failure_1 ⟨ordinary⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = NOT\_OK$
grd4: $failureStatus\_2 = OK$

then

act1: $var\_M\_15\_cho := 2$

end

Event failure_2 ⟨ordinary⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$

then

act1: $failureStatus\_2 := NOT\_OK$

end

Event treat_failure_2 ⟨ordinary⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = NOT\_OK$
grd4: $failureStatus\_1 = OK$

then

act1: $var\_M\_15\_cho := 1$

end

Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$

then

act1: $var\_M\_15\_cho := 0$

end

Event selection_twoWebsites ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$

then

act1: $var\_M\_15\_cho := 0$
end
Event selection ⟨convergent⟩ $\widehat{=}$
extends selection
any
$someCarts$
where
grd1: $var\_M\_1\_seq = 4$
grd2: $someCarts \subseteq SITES \times P$
grd3: $\mathrm{ran}(someCarts) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$
grd5: $var\_M\_15\_cho = 0$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := someCarts$
end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
when
grd1: $var\_M\_1\_seq = 3$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
when
grd1: $var\_M\_1\_seq = 2$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery
when
grd1: $var\_M\_1\_seq = 1$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
END

MACHINE M_151_
REFINES M_15_failure
SEES C_11_failure_status
VARIABLES

var_M_1_seq -
carts -
var_M_15_cho -
failureStatus_1 (one website)
failureStatus_2 (two websites)
var_M_151_par_A -
var_M_151_par_B -

INVARIANTS

type1: $var\_M\_151\_par\_A \in \mathbb{N}$
type2: $var\_M\_151\_par\_B \in \mathbb{N}$
DLF_3:
$$
\begin{aligned}
\neg((&var\_M\_1\_seq = 4 \\
&\land var\_M\_15\_cho = 1 \\
&\land failureStatus\_1 = OK) \\
\lor\ (&var\_M\_1\_seq = 4 \\
&\land var\_M\_15\_cho = 1 \\
&\land failureStatus\_1 = NOT\_OK \\
&\land failureStatus\_2 = OK) \\
\lor\ (&var\_M\_1\_seq = 4 \\
&\land var\_M\_15\_cho = 2 \\
&\land failureStatus\_2 = OK) \\
\lor\ (&var\_M\_1\_seq = 4 \\
&\land var\_M\_15\_cho = 2 \\
&\land failureStatus\_2 = NOT\_OK \\
&\land failureStatus\_1 = OK) \\
\lor\ (&var\_M\_1\_seq = 4 \\
&\land var\_M\_15\_cho = 1 \\
&\land failureStatus\_1 = OK) \\
\lor\ (&var\_M\_1\_seq = 4 \\
&\land var\_M\_15\_cho = 2 \\
&\land var\_M\_151\_par\_A = 1 \\
&\land failureStatus\_2 = OK) \\
\lor\ (&var\_M\_1\_seq = 4 \\
&\land var\_M\_15\_cho = 2 \\
&\land var\_M\_151\_par\_B = 1 \\
&\land failureStatus\_2 = OK) \\
\lor\ (&var\_M\_1\_seq = 4 \\
&\land var\_M\_15\_cho = 2 \\
&\land failureStatus\_2 = OK \\
&\land var\_M\_151\_par\_A = 0 \\
&\land var\_M\_151\_par\_B = 0) \\
\lor\ (&\exists someCarts \cdot \\
&\quad (var\_M\_1\_seq = 4 \\
&\quad \land someCarts \subseteq SITES \times P \\
&\quad \land \mathrm{ran}(someCarts) = P \\
&\quad \land (\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1) \\
&\quad \land var\_M\_15\_cho = 0)) \\
\lor\ &var\_M\_1\_seq = 3 \\
\lor\ &var\_M\_1\_seq = 2 \\
\lor\ &var\_M\_1\_seq = 1)
\end{aligned}
$$

$\Rightarrow$
$(var\_M\_1\_seq = 0$
$\lor (failureStatus\_1 = NOT\_OK \land failureStatus\_2 = NOT\_OK))$
deadlock => (finished or total failure)

VARIANT

$var\_M\_1\_seq + var\_M\_15\_cho + var\_M\_151\_par\_A + var\_M\_151\_par\_B$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$
act3: $carts := \varnothing$
act4: $var\_M\_15\_cho :\in \{1, 2\}$
act5: $failureStatus\_1 := OK$
act6: $failureStatus\_2 := OK$
act7: $var\_M\_151\_par\_A := 1$
act8: $var\_M\_151\_par\_B := 1$

end

Event failure_1 ⟨ordinary⟩ $\widehat{=}$

extends failure_1

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$

then

act1: $failureStatus\_1 := NOT\_OK$

end

Event treat_failure_1 ⟨ordinary⟩ $\widehat{=}$

extends treat_failure_1

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = NOT\_OK$
grd4: $failureStatus\_2 = OK$

then

act1: $var\_M\_15\_cho := 2$

end

Event failure_2 ⟨ordinary⟩ $\widehat{=}$

extends failure_2

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$

then

act1: $failureStatus\_2 := NOT\_OK$

end

Event treat_failure_2 ⟨ordinary⟩ $\widehat{=}$

extends treat_failure_2

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = NOT\_OK$
grd4: $failureStatus\_1 = OK$

then

act1: $var\_M\_15\_cho := 1$

end

Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$

extends selection_oneWebsite

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 1$

grd3: $failureStatus\_1 = OK$

then

act1: $var\_M\_15\_cho := 0$

end

Event selection_twoWebsites_A ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 2$

grd3: $var\_M\_151\_par\_A = 1$

grd4: $failureStatus\_2 = OK$

then

act1: $var\_M\_151\_par\_A := var\_M\_151\_par\_A - 1$

end

Event selection_twoWebsites_B ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 2$

grd3: $var\_M\_151\_par\_B = 1$

grd4: $failureStatus\_2 = OK$

then

act1: $var\_M\_151\_par\_B := var\_M\_151\_par\_B - 1$

end

Event selection_twoWebsites_join_A_B ⟨convergent⟩ $\widehat{=}$

extends selection_twoWebsites

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 2$

grd3: $failureStatus\_2 = OK$

grd4: $var\_M\_151\_par\_A = 0$

grd5: $var\_M\_151\_par\_B = 0$

then

act1: $var\_M\_15\_cho := 0$

end

Event selection ⟨convergent⟩ $\widehat{=}$

extends selection

any

$someCarts$

where

grd1: $var\_M\_1\_seq = 4$

grd2: $someCarts \subseteq SITES \times P$

grd3: $\mathrm{ran}(someCarts) = P$

grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$

grd5: $var\_M\_15\_cho = 0$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

act2: $carts := someCarts$

```
    end
Event payment ⟨convergent⟩ ≙
extends payment
    when
        grd1:  var_M_1_seq = 3
    then
        act1: var_M_1_seq := var_M_1_seq − 1
    end
Event billing ⟨convergent⟩ ≙
extends billing
    when
        grd1:  var_M_1_seq = 2
    then
        act1: var_M_1_seq := var_M_1_seq − 1
    end
Event delivery ⟨convergent⟩ ≙
extends delivery
    when
        grd1:  var_M_1_seq = 1
    then
        act1: var_M_1_seq := var_M_1_seq − 1
    end
END
```

MACHINE M_1511_reboot
REFINES M_151_
SEES C_11_failure_status
VARIABLES

var_M_1_seq -
carts -
var_M_15_cho -
failureStatus_1 (one website)
failureStatus_2 (two websites)
var_M_151_par_A -
var_M_151_par_B -
carts_ref -
var_M_1511_loop_1 -
var_M_1511_loop_2_A -
var_M_1511_loop_2_B -
site_1 -
site_2_A -
site_2_B -

INVARIANTS

type1: $carts\_ref \subseteq SITES \times P$
type2: $var\_M\_1511\_loop\_1 \in \mathbb{N}$
type3: $var\_M\_1511\_loop\_2\_A \in \mathbb{N}$
type4: $var\_M\_1511\_loop\_2\_B \in \mathbb{N}$
type5: $site\_1 \in SITES$
type6: $site\_2\_A \in SITES$
type7: $site\_2\_B \in SITES$
prop1: $var\_M\_15\_cho = 1 \Rightarrow \mathrm{dom}(carts\_ref) \subseteq \{site\_1\}$
prop2: $var\_M\_15\_cho = 2 \Rightarrow \mathrm{dom}(carts\_ref) \subseteq \{site\_2\_A, site\_2\_B\}$

VARIANT

$var\_M\_1\_seq + var\_M\_15\_cho + var\_M\_151\_par\_A + var\_M\_151\_par\_B + var\_M\_1511\_loop\_1 + var\_M\_1511\_loop\_2\_A + var\_M\_1511\_loop\_2\_B$

EVENTS

Initialisation

begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_1511\_loop\_1, var\_M\_1511\_loop\_2\_A, var\_M\_1511\_loop\_2\_B :|$
$var\_M\_1511\_loop\_1' = \mathrm{card}(P)$
$\wedge\ var\_M\_1511\_loop\_2\_A' + var\_M\_1511\_loop\_2\_B' = \mathrm{card}(P)$
$\wedge\ var\_M\_1511\_loop\_2\_A' \in \mathbb{N}$
$\wedge\ var\_M\_1511\_loop\_2\_B' \in \mathbb{N}$
act3: $carts := \varnothing$
act4: $var\_M\_15\_cho :\in \{1, 2\}$
act5: $failureStatus\_1 := OK$
act6: $failureStatus\_2 := OK$
act7: $var\_M\_151\_par\_A := 1$
act8: $var\_M\_151\_par\_B := 1$
act9: $carts\_ref := \varnothing$
act10: $site\_1 :\in SITES$
act11: $site\_2\_A :\in SITES$
act12: $site\_2\_B :\in SITES$

end
Event failure_1 ⟨ordinary⟩ $\widehat{=}$
extends failure_1
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
then
act1: $failureStatus\_1 := NOT\_OK$
end
Event treat_failure_1 ⟨ordinary⟩ $\widehat{=}$
extends treat_failure_1
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = NOT\_OK$
grd4: $failureStatus\_2 = OK$
then
act1: $var\_M\_15\_cho := 2$
act2: $carts\_ref := \varnothing$
carts_ref is reinitialized to rebuild the initial state
end
Event failure_2 ⟨ordinary⟩ $\widehat{=}$
extends failure_2
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
then
act1: $failureStatus\_2 := NOT\_OK$
end
Event treat_failure_2 ⟨ordinary⟩ $\widehat{=}$
extends treat_failure_2
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = NOT\_OK$
grd4: $failureStatus\_1 = OK$
then
act1: $var\_M\_15\_cho := 1$
act2: $carts\_ref := \varnothing$
carts_ref is reinitialized to rebuild the initial state
end
Event selection_oneWebsite_loop ⟨convergent⟩ $\widehat{=}$
any
someProduct
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
grd4: $var\_M\_1511\_loop\_1 > 0$
grd5: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$
then

act1: $var\_M\_1511\_loop\_1 := var\_M\_1511\_loop\_1 - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_1 \mapsto someProduct\}$
end
Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$
extends selection_oneWebsite
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
grd4: $var\_M\_1511\_loop\_1 = 0$
then
act1: $var\_M\_15\_cho := 0$
end
Event selection_twoWebsites_A_loop ⟨convergent⟩ $\widehat{=}$
any
someProduct
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_A = 1$
grd5: $var\_M\_1511\_loop\_2\_A > 0$
grd6: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$
then
act1: $var\_M\_1511\_loop\_2\_A := var\_M\_1511\_loop\_2\_A - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_2\_A \mapsto someProduct\}$
end
Event selection_twoWebsites_A ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_A
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $var\_M\_151\_par\_A = 1$
grd4: $failureStatus\_2 = OK$
grd5: $var\_M\_1511\_loop\_2\_A = 0$
then
act1: $var\_M\_151\_par\_A := var\_M\_151\_par\_A - 1$
end
Event selection_twoWebsites_B_loop ⟨convergent⟩ $\widehat{=}$
any
someProduct
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_B = 1$
grd5: $var\_M\_1511\_loop\_2\_B > 0$
grd6: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$
then
act1: $var\_M\_1511\_loop\_2\_B := var\_M\_1511\_loop\_2\_B - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_2\_B \mapsto someProduct\}$
end
Event selection_twoWebsites_B ⟨convergent⟩ $\widehat{=}$

extends selection_twoWebsites_B
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_15\_cho = 2$
    grd3: $var\_M\_151\_par\_B = 1$
    grd4: $failureStatus\_2 = OK$
    grd5: $var\_M\_1511\_loop\_2\_B = 0$
  then
    act1: $var\_M\_151\_par\_B := var\_M\_151\_par\_B - 1$
  end
Event selection_twoWebsites_join_A_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_join_A_B
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_15\_cho = 2$
    grd3: $failureStatus\_2 = OK$
    grd4: $var\_M\_151\_par\_A = 0$
    grd5: $var\_M\_151\_par\_B = 0$
  then
    act1: $var\_M\_15\_cho := 0$
  end
Event confirmSelection ⟨convergent⟩ $\widehat{=}$
refines selection
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $\mathrm{ran}(carts\_ref) = P$
    grd3: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow \mathrm{card}(carts\_ref^{-1}[\{p\}]) = 1$
    grd4: $var\_M\_15\_cho = 0$
  with
    someCarts: $someCarts = carts\_ref$
  then
    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
    act2: $carts := carts\_ref$
  end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
  when
    grd1: $var\_M\_1\_seq = 3$
  then
    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
  end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
  when
    grd1: $var\_M\_1\_seq = 2$
  then
    act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
  end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery
  when
    grd1: $var\_M\_1\_seq = 1$
  then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

END

MACHINE M_15111_
REFINES M_1511_reboot
SEES C_11_failure_status
VARIABLES

var_M_1_seq -
carts -
var_M_15_cho -
failureStatus_1 (one website)
failureStatus_2 (two websites)
var_M_151_par_A -
var_M_151_par_B -
carts_ref -
var_M_1511_loop_1 -
var_M_1511_loop_2_A -
var_M_1511_loop_2_B -
site_1 -
site_2_A -
site_2_B -
var_M_15111_seq_1 -
var_M_15111_seq_2_A -
var_M_15111_seq_2_B -
selectedItem_1 -
selectedItem_2_A -
selectedItem_2_B -

INVARIANTS

type1: $var\_M\_15111\_seq\_1 \in \mathbb{N}$
type2: $var\_M\_15111\_seq\_2\_A \in \mathbb{N}$
type3: $var\_M\_15111\_seq\_2\_B \in \mathbb{N}$
type4: $selectedItem\_1 \in \mathbb{P}(P)$
type5: $selectedItem\_2\_A \in \mathbb{P}(P)$
type6: $selectedItem\_2\_B \in \mathbb{P}(P)$
prop1: $var\_M\_15111\_seq\_1 \geq 1 \Rightarrow \mathrm{card}(selectedItem\_1) = 0$
prop2: $var\_M\_15111\_seq\_1 < 1 \Rightarrow \mathrm{card}(selectedItem\_1) = 1$
prop3: $var\_M\_15111\_seq\_2\_A \geq 1 \Rightarrow \mathrm{card}(selectedItem\_2\_A) = 0$
prop4: $var\_M\_15111\_seq\_2\_A < 1 \Rightarrow \mathrm{card}(selectedItem\_2\_A) = 1$
prop5: $var\_M\_15111\_seq\_2\_B \geq 1 \Rightarrow \mathrm{card}(selectedItem\_2\_B) = 0$
prop6: $var\_M\_15111\_seq\_2\_B < 1 \Rightarrow \mathrm{card}(selectedItem\_2\_B) = 1$

VARIANT

$var\_M\_1\_seq + var\_M\_15\_cho + var\_M\_151\_par\_A + var\_M\_151\_par\_B + var\_M\_1511\_loop\_1 + var\_M\_1511\_loop\_2\_A + var\_M\_1511\_loop\_2\_B + var\_M\_15111\_seq\_1 + var\_M\_15111\_seq\_2\_A + var\_M\_15111\_seq\_2\_B$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_1511\_loop\_1, var\_M\_1511\_loop\_2\_A, var\_M\_1511\_loop\_2\_B :|$
$var\_M\_1511\_loop\_1' = \mathrm{card}(P)$
$\land\ var\_M\_1511\_loop\_2\_A' + var\_M\_1511\_loop\_2\_B' = \mathrm{card}(P)$
$\land\ var\_M\_1511\_loop\_2\_A' \in \mathbb{N}$
$\land\ var\_M\_1511\_loop\_2\_B' \in \mathbb{N}$

act3: $carts := \varnothing$
act4: $var\_M\_15\_cho :\in \{1, 2\}$
act5: $failureStatus\_1 := OK$
act6: $failureStatus\_2 := OK$
act7: $var\_M\_151\_par\_A := 1$
act8: $var\_M\_151\_par\_B := 1$
act9: $carts\_ref := \varnothing$
act10: $site\_1 :\in SITES$
act11: $site\_2\_A :\in SITES$
act12: $site\_2\_B :\in SITES$
act13: $var\_M\_15111\_seq\_1 := 1$
act14: $selectedItem\_1 := \varnothing$
act15: $var\_M\_15111\_seq\_2\_A := 1$
act16: $selectedItem\_2\_A := \varnothing$
act17: $var\_M\_15111\_seq\_2\_B := 1$
act18: $selectedItem\_2\_B := \varnothing$

end

Event failure_1 ⟨ordinary⟩ $\widehat{=}$
extends failure_1

when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$

then
act1: $failureStatus\_1 := NOT\_OK$

end

Event treat_failure_1 ⟨ordinary⟩ $\widehat{=}$
extends treat_failure_1

when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = NOT\_OK$
grd4: $failureStatus\_2 = OK$

then
act1: $var\_M\_15\_cho := 2$
act2: $carts\_ref := \varnothing$
carts_ref is reinitialized to rebuild the initial state

end

Event failure_2 ⟨ordinary⟩ $\widehat{=}$
extends failure_2

when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$

then
act1: $failureStatus\_2 := NOT\_OK$

end

Event treat_failure_2 ⟨ordinary⟩ $\widehat{=}$
extends treat_failure_2

when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = NOT\_OK$

grd4: $failureStatus\_1 = OK$
then
act1: $var\_M\_15\_cho := 1$
act2: $carts\_ref := \varnothing$
carts_ref is reinitialized to rebuild the initial state
end
Event selectItemInItemList_1 ⟨convergent⟩ $\widehat{=}$
any
someProduct
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
grd4: $var\_M\_1511\_loop\_1 > 0$
grd5: $var\_M\_15111\_seq\_1 = 1$
grd6: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$
then
act1: $var\_M\_15111\_seq\_1 := var\_M\_15111\_seq\_1 - 1$
act2: $selectedItem\_1 := \{someProduct\}$
end
Event addSelectedItemToCart_1 ⟨convergent⟩ $\widehat{=}$
refines selection_oneWebsite_loop
any
item used to access the element in selectedItem_1
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
grd4: $var\_M\_1511\_loop\_1 > 0$
grd5: $var\_M\_15111\_seq\_1 = 0$
grd6: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_1 = \{p\}$
grd7: $selectedItem\_1 = \{item\}$
with
someProduct: $selectedItem\_1 = \{someProduct\}$
then
act1: $var\_M\_1511\_loop\_1 := var\_M\_1511\_loop\_1 - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_1 \mapsto item\}$
end
Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$
extends selection_oneWebsite
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
grd4: $var\_M\_1511\_loop\_1 = 0$
then
act1: $var\_M\_15\_cho := 0$
end
Event selectItemInItemList_2_A ⟨convergent⟩ $\widehat{=}$
any
someProduct
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$

grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_A = 1$
grd5: $var\_M\_1511\_loop\_2\_A > 0$
grd6: $var\_M\_15111\_seq\_2\_A = 1$
grd7: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_15111\_seq\_2\_A := var\_M\_15111\_seq\_2\_A - 1$
act2: $selectedItem\_2\_A := \{someProduct\}$

end

Event addSelectedItemToCart_2_A ⟨convergent⟩ $\widehat{=}$
refines selection_twoWebsites_A_loop

any

item used to access the element in selectedItem_2_A

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_A = 1$
grd5: $var\_M\_1511\_loop\_2\_A > 0$
grd6: $var\_M\_15111\_seq\_2\_A = 0$
grd7: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_2\_A = \{p\}$
grd8: $selectedItem\_2\_A = \{item\}$

with

someProduct: $selectedItem\_2\_A = \{someProduct\}$

then

act1: $var\_M\_1511\_loop\_2\_A := var\_M\_1511\_loop\_2\_A - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_2\_A \mapsto item\}$

end

Event selection_twoWebsites_A ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_A

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $var\_M\_151\_par\_A = 1$
grd4: $failureStatus\_2 = OK$
grd5: $var\_M\_1511\_loop\_2\_A = 0$

then

act1: $var\_M\_151\_par\_A := var\_M\_151\_par\_A - 1$

end

Event selectItemInItemList_2_B ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_B = 1$
grd5: $var\_M\_1511\_loop\_2\_B > 0$
grd6: $var\_M\_15111\_seq\_2\_B = 1$
grd7: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_15111\_seq\_2\_B := var\_M\_15111\_seq\_2\_B - 1$
act2: $selectedItem\_2\_B := \{someProduct\}$

end

Event addSelectedItemToCart_2_B ⟨convergent⟩ $\widehat{=}$
refines selection_twoWebsites_B_loop
any
item used to access the element in selectedItem_2_B
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_B = 1$
grd5: $var\_M\_1511\_loop\_2\_B > 0$
grd6: $var\_M\_15111\_seq\_2\_B = 0$
grd7: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \land selectedItem\_2\_B = \{p\}$
grd8: $selectedItem\_2\_B = \{item\}$
with
someProduct: $selectedItem\_2\_B = \{someProduct\}$
then
act1: $var\_M\_1511\_loop\_2\_B := var\_M\_1511\_loop\_2\_B - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_2\_B \mapsto item\}$
end
Event selection_twoWebsites_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_B
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $var\_M\_151\_par\_B = 1$
grd4: $failureStatus\_2 = OK$
grd5: $var\_M\_1511\_loop\_2\_B = 0$
then
act1: $var\_M\_151\_par\_B := var\_M\_151\_par\_B - 1$
end
Event selection_twoWebsites_join_A_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_join_A_B
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_A = 0$
grd5: $var\_M\_151\_par\_B = 0$
then
act1: $var\_M\_15\_cho := 0$
end
Event confirmSelection ⟨convergent⟩ $\widehat{=}$
extends confirmSelection
when
grd1: $var\_M\_1\_seq = 4$
grd2: $\mathrm{ran}(carts\_ref) = P$
grd3: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow \mathrm{card}(carts\_ref^{-1}[\{p\}]) = 1$
grd4: $var\_M\_15\_cho = 0$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := carts\_ref$
end
Event payment ⟨convergent⟩ $\widehat{=}$

```
extends payment
      when
            grd1:  var_M_1_seq = 3
      then
            act1: var_M_1_seq := var_M_1_seq − 1
      end
Event billing ⟨convergent⟩ ≙
extends billing
      when
            grd1:  var_M_1_seq = 2
      then
            act1: var_M_1_seq := var_M_1_seq − 1
      end
Event delivery ⟨convergent⟩ ≙
extends delivery
      when
            grd1:  var_M_1_seq = 1
      then
            act1: var_M_1_seq := var_M_1_seq − 1
      end
END
```

MACHINE M_1512_repair
REFINES M_151_
SEES C_11_failure_status
VARIABLES

var_M_1_seq -
carts -
var_M_15_cho -
failureStatus_1 (one website)
failureStatus_2 (two websites)
var_M_151_par_A -
var_M_151_par_B -
carts_ref -
var_M_1511_loop_1 -
var_M_1511_loop_2_A -
var_M_1511_loop_2_B -
site_1 -
site_2_A -
site_2_B -

INVARIANTS

type1: $carts\_ref \subseteq SITES \times P$
type2: $var\_M\_1511\_loop\_1 \in \mathbb{N}$
type3: $var\_M\_1511\_loop\_2\_A \in \mathbb{N}$
type4: $var\_M\_1511\_loop\_2\_B \in \mathbb{N}$
type5: $site\_1 \in SITES$
type6: $site\_2\_A \in SITES$
type7: $site\_2\_B \in SITES$
prop1: $var\_M\_15\_cho = 1 \Rightarrow \mathrm{dom}(carts\_ref) \subseteq \{site\_1\}$
prop2: $var\_M\_15\_cho = 2 \Rightarrow \mathrm{dom}(carts\_ref) \subseteq \{site\_2\_A, site\_2\_B\}$
prop3: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow \mathrm{card}(carts\_ref^{-1}[\{p\}]) = 1$
tech1: $var\_M\_15\_cho = 1 \Rightarrow \mathrm{card}(P) - \mathrm{card}(\mathrm{ran}(carts\_ref)) = var\_M\_1511\_loop\_1$
tech2: $var\_M\_15\_cho = 2 \Rightarrow \mathrm{card}(P) - \mathrm{card}(\mathrm{ran}(carts\_ref)) = var\_M\_1511\_loop\_2\_A + var\_M\_1511\_loop\_2\_B$
thm1: ⟨theorem⟩ $\forall A, B, e \cdot (\mathrm{finite}(A) \wedge \mathrm{finite}(B) \wedge A \subseteq PRODUCTS \wedge B \subseteq A \wedge \mathrm{card}(A) - \mathrm{card}(B) - 1 = 0 \wedge e \in A \setminus B) \Rightarrow B \cup \{e\} = A$
tech3: $(var\_M\_15\_cho = 1 \wedge var\_M\_1511\_loop\_1 = 0) \Rightarrow \mathrm{ran}(carts\_ref) = P$
tech4: $(var\_M\_15\_cho = 2 \wedge var\_M\_1511\_loop\_2\_A = 0 \wedge var\_M\_1511\_loop\_2\_B = 0) \Rightarrow \mathrm{ran}(carts\_ref) = P$
DLF_4: $\neg($

$($
$var\_M\_1\_seq = 4$
$\wedge\, var\_M\_15\_cho = 1$
$\wedge\, failureStatus\_1 = OK$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\, var\_M\_15\_cho = 1$
$\wedge\, failureStatus\_1 = NOT\_OK$
$\wedge\, failureStatus\_2 = OK$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\, var\_M\_15\_cho = 2$

$\wedge\ failureStatus\_2 = OK$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = NOT\_OK$
$\wedge\ failureStatus\_1 = OK$
$) \vee ($
$\exists someProduct\cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 1$
$\wedge\ failureStatus\_1 = OK$
$\wedge\ var\_M\_1511\_loop\_1 > 0$
$\wedge\ someProduct \in P \setminus \mathrm{ran}(carts\_ref))$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 1$
$\wedge\ failureStatus\_1 = OK$
$\wedge\ var\_M\_1511\_loop\_1 = 0$
$) \vee ($
$\exists someProduct\cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_151\_par\_A = 1$
$\wedge\ var\_M\_1511\_loop\_2\_A > 0$
$\wedge\ someProduct \in P \setminus \mathrm{ran}(carts\_ref))$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ var\_M\_151\_par\_A = 1$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_1511\_loop\_2\_A = 0$
$) \vee ($
$\exists someProduct\cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_151\_par\_B = 1$
$\wedge\ var\_M\_1511\_loop\_2\_B > 0$
$\wedge\ someProduct \in P \setminus \mathrm{ran}(carts\_ref))$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ var\_M\_151\_par\_B = 1$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_1511\_loop\_2\_B = 0$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_151\_par\_A = 0$
$\wedge\ var\_M\_151\_par\_B = 0$
$\wedge\ var\_M\_1511\_loop\_2\_A = 0$
$\wedge\ var\_M\_1511\_loop\_2\_B = 0$
$) \vee ($

$var\_M\_1\_seq = 4$
$\wedge$ ran$(carts_ref) = P$
$\wedge$ $(\forall p \cdot p \in$ ran$(carts_ref) \Rightarrow$ card$(carts_ref^{-1}[\{p\}]) = 1)$
$\wedge$ $var\_M\_15\_cho = 0$
$) \vee ($
$var\_M\_1\_seq = 3$
$) \vee ($
$var\_M\_1\_seq = 2$
$) \vee ($
$var\_M\_1\_seq = 1$
$))$
$\Rightarrow$
$(var\_M\_1\_seq = 0$
$\vee$ $(failureStatus\_1 = NOT\_OK \wedge failureStatus\_2 = NOT\_OK))$

deadlock => (finished or total failure)

VARIANT

$var\_M\_1\_seq + var\_M\_15\_cho + var\_M\_151\_par\_A + var\_M\_151\_par\_B + var\_M\_1511\_loop\_1 + var\_M\_1511\_loop\_2\_A + var\_M\_1511\_loop\_2\_B$

EVENTS

Initialisation

begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_1511\_loop\_1, var\_M\_1511\_loop\_2\_A, var\_M\_1511\_loop\_2\_B :|$
$var\_M\_1511\_loop\_1' =$ card(P)
$\wedge$ $var\_M\_1511\_loop\_2\_A' + var\_M\_1511\_loop\_2\_B' =$ card(P)
$\wedge$ $var\_M\_1511\_loop\_2\_A' \in \mathbb{N}$
$\wedge$ $var\_M\_1511\_loop\_2\_B' \in \mathbb{N}$
act3: $carts := \varnothing$
act4: $var\_M\_15\_cho :\in \{1, 2\}$
act5: $failureStatus\_1 := OK$
act6: $failureStatus\_2 := OK$
act7: $var\_M\_151\_par\_A := 1$
act8: $var\_M\_151\_par\_B := 1$
act9: $carts\_ref := \varnothing$
act10: $site\_1 :\in SITES$
act11: $site\_2\_A :\in SITES$
act12: $site\_2\_B :\in SITES$

end

Event failure_1 ⟨ordinary⟩ $\widehat{=}$

extends failure_1

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$

then

act1: $failureStatus\_1 := NOT\_OK$

end

Event treat_failure_1 ⟨ordinary⟩ $\widehat{=}$

extends treat_failure_1

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = NOT\_OK$

grd4: $failureStatus\_2 = OK$

then

act1: $var\_M\_15\_cho := 2$

act2: $carts\_ref := \{p \cdot p \in \operatorname{ran}(carts\_ref) | site\_2\_A \mapsto p\}$

act3: $var\_M\_1511\_loop\_2\_A, var\_M\_1511\_loop\_2\_B :|$
$var\_M\_1511\_loop\_2\_A' + var\_M\_1511\_loop\_2\_B' = var\_M\_1511\_loop\_1$
$\land\ var\_M\_1511\_loop\_2\_A' \in \mathbb{N}$
$\land\ var\_M\_1511\_loop\_2\_B' \in \mathbb{N}$

end

Event failure_2 ⟨ordinary⟩ $\widehat{=}$

extends failure_2

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 2$

grd3: $failureStatus\_2 = OK$

then

act1: $failureStatus\_2 := NOT\_OK$

end

Event treat_failure_2 ⟨ordinary⟩ $\widehat{=}$

extends treat_failure_2

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 2$

grd3: $failureStatus\_2 = NOT\_OK$

grd4: $failureStatus\_1 = OK$

then

act1: $var\_M\_15\_cho := 1$

act2: $carts\_ref := \{p \cdot p \in \operatorname{ran}(carts\_ref) | site\_1 \mapsto p\}$

act3: $var\_M\_1511\_loop\_1 := var\_M\_1511\_loop\_2\_A + var\_M\_1511\_loop\_2\_B$

end

Event selection_oneWebsite_loop ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 1$

grd3: $failureStatus\_1 = OK$

grd4: $var\_M\_1511\_loop\_1 > 0$

grd5: $someProduct \in P \setminus \operatorname{ran}(carts\_ref)$

then

act1: $var\_M\_1511\_loop\_1 := var\_M\_1511\_loop\_1 - 1$

act2: $carts\_ref := carts\_ref \cup \{site\_1 \mapsto someProduct\}$

end

Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$

extends selection_oneWebsite

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 1$

grd3: $failureStatus\_1 = OK$

grd4: $var\_M\_1511\_loop\_1 = 0$

then

act1: $var\_M\_15\_cho := 0$

end

Event selection_twoWebsites_A_loop ⟨convergent⟩ $\widehat{=}$
  any
    someProduct
  where
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_15\_cho = 2$
    grd3: $failureStatus\_2 = OK$
    grd4: $var\_M\_151\_par\_A = 1$
    grd5: $var\_M\_1511\_loop\_2\_A > 0$
    grd6: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$
  then
    act1: $var\_M\_1511\_loop\_2\_A := var\_M\_1511\_loop\_2\_A - 1$
    act2: $carts\_ref := carts\_ref \cup \{site\_2\_A \mapsto someProduct\}$
  end
Event selection_twoWebsites_A ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_A
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_15\_cho = 2$
    grd3: $var\_M\_151\_par\_A = 1$
    grd4: $failureStatus\_2 = OK$
    grd5: $var\_M\_1511\_loop\_2\_A = 0$
  then
    act1: $var\_M\_151\_par\_A := var\_M\_151\_par\_A - 1$
  end
Event selection_twoWebsites_B_loop ⟨convergent⟩ $\widehat{=}$
  any
    someProduct
  where
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_15\_cho = 2$
    grd3: $failureStatus\_2 = OK$
    grd4: $var\_M\_151\_par\_B = 1$
    grd5: $var\_M\_1511\_loop\_2\_B > 0$
    grd6: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$
  then
    act1: $var\_M\_1511\_loop\_2\_B := var\_M\_1511\_loop\_2\_B - 1$
    act2: $carts\_ref := carts\_ref \cup \{site\_2\_B \mapsto someProduct\}$
  end
Event selection_twoWebsites_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_B
  when
    grd1: $var\_M\_1\_seq = 4$
    grd2: $var\_M\_15\_cho = 2$
    grd3: $var\_M\_151\_par\_B = 1$
    grd4: $failureStatus\_2 = OK$
    grd5: $var\_M\_1511\_loop\_2\_B = 0$
  then
    act1: $var\_M\_151\_par\_B := var\_M\_151\_par\_B - 1$
  end
Event selection_twoWebsites_join_A_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_join_A_B
  when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_A = 0$
grd5: $var\_M\_151\_par\_B = 0$
grd6: $var\_M\_1511\_loop\_2\_A = 0$
grd7: $var\_M\_1511\_loop\_2\_B = 0$

then

act1: $var\_M\_15\_cho := 0$

end

Event confirmSelection ⟨convergent⟩ $\widehat{=}$
refines selection

when

grd1: $var\_M\_1\_seq = 4$
grd3: $\mathrm{ran}(carts\_ref) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow \mathrm{card}(carts\_ref^{-1}[\{p\}]) = 1$
grd5: $var\_M\_15\_cho = 0$

with

someCarts: $someCarts = carts\_ref$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := carts\_ref$

end

Event payment ⟨convergent⟩ $\widehat{=}$
extends payment

when

grd1: $var\_M\_1\_seq = 3$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event billing ⟨convergent⟩ $\widehat{=}$
extends billing

when

grd1: $var\_M\_1\_seq = 2$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery

when

grd1: $var\_M\_1\_seq = 1$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

END

MACHINE M_15121_
REFINES M_1512_repair
SEES C_11_failure_status
VARIABLES

var_M_1_seq -
carts -
var_M_15_cho -
failureStatus_1 (one website)
failureStatus_2 (two websites)
var_M_151_par_A -
var_M_151_par_B -
carts_ref -
var_M_1511_loop_1 -
var_M_1511_loop_2_A -
var_M_1511_loop_2_B -
site_1 -
site_2_A -
site_2_B -
var_M_15111_seq_1 -
var_M_15111_seq_2_A -
var_M_15111_seq_2_B -
selectedItem_1 -
selectedItem_2_A -
selectedItem_2_B -

INVARIANTS

type1: $var\_M\_15111\_seq\_1 \in \mathbb{N}$
type2: $var\_M\_15111\_seq\_2\_A \in \mathbb{N}$
type3: $var\_M\_15111\_seq\_2\_B \in \mathbb{N}$
type4: $selectedItem\_1 \in \mathbb{P}(P)$
type5: $selectedItem\_2\_A \in \mathbb{P}(P)$
type6: $selectedItem\_2\_B \in \mathbb{P}(P)$
prop1: $var\_M\_15111\_seq\_1 \geq 1 \Rightarrow \mathrm{card}(selectedItem\_1) = 0$
prop2: $var\_M\_15111\_seq\_1 < 1 \Rightarrow \mathrm{card}(selectedItem\_1) = 1$
prop3: $var\_M\_15111\_seq\_2\_A \geq 1 \Rightarrow \mathrm{card}(selectedItem\_2\_A) = 0$
prop4: $var\_M\_15111\_seq\_2\_A < 1 \Rightarrow \mathrm{card}(selectedItem\_2\_A) = 1$
prop5: $var\_M\_15111\_seq\_2\_B \geq 1 \Rightarrow \mathrm{card}(selectedItem\_2\_B) = 0$
prop6: $var\_M\_15111\_seq\_2\_B < 1 \Rightarrow \mathrm{card}(selectedItem\_2\_B) = 1$
DLF_5: $\neg($
$($
$var\_M\_1\_seq = 4$
$\wedge\, var\_M\_15\_cho = 1$
$\wedge\, failureStatus\_1 = OK$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\, var\_M\_15\_cho = 1$
$\wedge\, failureStatus\_1 = NOT\_OK$
$\wedge\, failureStatus\_2 = OK$
$) \vee ($
$var\_M\_1\_seq = 4$

$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = NOT\_OK$
$\wedge\ failureStatus\_1 = OK$
$\wedge\ var\_M\_15111\_seq\_2\_A = 0$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = NOT\_OK$
$\wedge\ failureStatus\_1 = OK$
$\wedge\ var\_M\_15111\_seq\_2\_A \neq 0$
$\wedge\ var\_M\_15111\_seq\_2\_B = 0$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = NOT\_OK$
$\wedge\ failureStatus\_1 = OK$
$\wedge\ var\_M\_15111\_seq\_2\_A \neq 0$
$\wedge\ var\_M\_15111\_seq\_2\_B \neq 0$
$) \vee ($
$\exists someProduct \cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 1$
$\wedge\ failureStatus\_1 = OK$
$\wedge\ var\_M\_1511\_loop\_1 > 0$
$\wedge\ var\_M\_15111\_seq\_1 = 1$
$\wedge\ someProduct \in P \setminus \mathrm{ran}(carts\_ref))$
$) \vee ($
$\exists item \cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 1$
$\wedge\ failureStatus\_1 = OK$
$\wedge\ var\_M\_1511\_loop\_1 > 0$
$\wedge\ var\_M\_15111\_seq\_1 = 0$
$\wedge\ (\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_1 = \{p\})$
$\wedge\ selectedItem\_1 = \{item\})$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 1$
$\wedge\ failureStatus\_1 = OK$
$\wedge\ var\_M\_1511\_loop\_1 = 0$
$) \vee ($
$\exists someProduct \cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_151\_par\_A = 1$
$\wedge\ var\_M\_1511\_loop\_2\_A > 0$
$\wedge\ var\_M\_15111\_seq\_2\_A = 1$
$\wedge\ someProduct \in P \setminus \mathrm{ran}(carts\_ref))$
$) \vee ($
$\exists item \cdot$

$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_151\_par\_A = 1$
$\wedge\ var\_M\_1511\_loop\_2\_A > 0$
$\wedge\ var\_M\_15111\_seq\_2\_A = 0$
$\wedge\ (\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_2\_A = \{p\})$
$\wedge\ selectedItem\_2\_A = \{item\})$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ var\_M\_151\_par\_A = 1$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_1511\_loop\_2\_A = 0$
$) \vee ($
$\exists someProduct \cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_151\_par\_B = 1$
$\wedge\ var\_M\_1511\_loop\_2\_B > 0$
$\wedge\ var\_M\_15111\_seq\_2\_B = 1$
$\wedge\ someProduct \in P \setminus \mathrm{ran}(carts\_ref))$
$) \vee ($
$\exists item \cdot$
$(var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_151\_par\_B = 1$
$\wedge\ var\_M\_1511\_loop\_2\_B > 0$
$\wedge\ var\_M\_15111\_seq\_2\_B = 0$
$\wedge\ (\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_2\_B = \{p\})$
$\wedge\ selectedItem\_2\_B = \{item\})$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ var\_M\_151\_par\_B = 1$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_1511\_loop\_2\_B = 0$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ var\_M\_15\_cho = 2$
$\wedge\ failureStatus\_2 = OK$
$\wedge\ var\_M\_151\_par\_A = 0$
$\wedge\ var\_M\_151\_par\_B = 0$
$\wedge\ var\_M\_1511\_loop\_2\_A = 0$
$\wedge\ var\_M\_1511\_loop\_2\_B = 0$
$) \vee ($
$var\_M\_1\_seq = 4$
$\wedge\ \mathrm{ran}(carts\_ref) = P$
$\wedge\ (\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow \mathrm{card}(carts\_ref^{-1}[\{p\}]) = 1)$
$\wedge\ var\_M\_15\_cho = 0$
$) \vee ($
$var\_M\_1\_seq = 3$
$) \vee ($

$var\_M\_1\_seq = 2$
$) \vee ($
$var\_M\_1\_seq = 1$
$))$
$\Rightarrow$
$(var\_M\_1\_seq = 0$
$\vee (failureStatus\_1 = NOT\_OK \wedge failureStatus\_2 = NOT\_OK))$

deadlock => (finished or total failure)

VARIANT

$var\_M\_1\_seq+var\_M\_15\_cho+var\_M\_151\_par\_A+var\_M\_151\_par\_B+var\_M\_1511\_loop\_1+ var\_M\_1511\_loop\_2\_A+var\_M\_1511\_loop\_2\_B+var\_M\_15111\_seq\_1+var\_M\_15111\_seq\_2\_A+ var\_M\_15111\_seq\_2\_B$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$
act2: $var\_M\_1511\_loop\_1, var\_M\_1511\_loop\_2\_A, var\_M\_1511\_loop\_2\_B :|$
$var\_M\_1511\_loop\_1' = \mathrm{card}(P)$
$\wedge\ var\_M\_1511\_loop\_2\_A' + var\_M\_1511\_loop\_2\_B' = \mathrm{card}(P)$
$\wedge\ var\_M\_1511\_loop\_2\_A' \in \mathbb{N}$
$\wedge\ var\_M\_1511\_loop\_2\_B' \in \mathbb{N}$
act3: $carts := \varnothing$
act4: $var\_M\_15\_cho :\in \{1, 2\}$
act5: $failureStatus\_1 := OK$
act6: $failureStatus\_2 := OK$
act7: $var\_M\_151\_par\_A := 1$
act8: $var\_M\_151\_par\_B := 1$
act9: $carts\_ref := \varnothing$
act10: $site\_1 :\in SITES$
act11: $site\_2\_A :\in SITES$
act12: $site\_2\_B :\in SITES$
act14: $var\_M\_15111\_seq\_1 := 1$
act15: $selectedItem\_1 := \varnothing$
act16: $var\_M\_15111\_seq\_2\_A := 1$
act17: $selectedItem\_2\_A := \varnothing$
act18: $var\_M\_15111\_seq\_2\_B := 1$
act19: $selectedItem\_2\_B := \varnothing$

end

Event failure_1 ⟨ordinary⟩ $\widehat{=}$

extends failure_1

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$

then

act1: $failureStatus\_1 := NOT\_OK$

end

Event treat_failure_1 ⟨ordinary⟩ $\widehat{=}$

extends treat_failure_1

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = NOT\_OK$

grd4: $failureStatus\_2 = OK$

then

act1: $var\_M\_15\_cho := 2$

act2: $carts\_ref := \{p \cdot p \in \mathrm{ran}(carts\_ref) | site\_2\_A \mapsto p\}$

act3: $var\_M\_1511\_loop\_2\_A, var\_M\_1511\_loop\_2\_B :|$
$var\_M\_1511\_loop\_2\_A' + var\_M\_1511\_loop\_2\_B' = var\_M\_1511\_loop\_1$
$\wedge\ var\_M\_1511\_loop\_2\_A' \in \mathbb{N}$
$\wedge\ var\_M\_1511\_loop\_2\_B' \in \mathbb{N}$

act4: $var\_M\_15111\_seq\_2\_A := var\_M\_15111\_seq\_1$

act5: $selectedItem\_2\_A := selectedItem\_1$

end

Event failure_2 ⟨ordinary⟩ $\widehat{=}$

extends failure_2

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 2$

grd3: $failureStatus\_2 = OK$

then

act1: $failureStatus\_2 := NOT\_OK$

end

Event treat_failure_2_0 ⟨ordinary⟩ $\widehat{=}$

extends treat_failure_2

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 2$

grd3: $failureStatus\_2 = NOT\_OK$

grd4: $failureStatus\_1 = OK$

grd5: $var\_M\_15111\_seq\_2\_A = 0$

then

act1: $var\_M\_15\_cho := 1$

act2: $carts\_ref := \{p \cdot p \in \mathrm{ran}(carts\_ref) | site\_1 \mapsto p\}$

act3: $var\_M\_1511\_loop\_1 := var\_M\_1511\_loop\_2\_A + var\_M\_1511\_loop\_2\_B$

act4: $var\_M\_15111\_seq\_1 := var\_M\_15111\_seq\_2\_A$

act5: $selectedItem\_1 := selectedItem\_2\_A$

end

Event treat_failure_2_1 ⟨ordinary⟩ $\widehat{=}$

extends treat_failure_2

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_15\_cho = 2$

grd3: $failureStatus\_2 = NOT\_OK$

grd4: $failureStatus\_1 = OK$

grd5: $var\_M\_15111\_seq\_2\_A \neq 0$

grd6: $var\_M\_15111\_seq\_2\_B = 0$

then

act1: $var\_M\_15\_cho := 1$

act2: $carts\_ref := \{p \cdot p \in \mathrm{ran}(carts\_ref) | site\_1 \mapsto p\}$

act3: $var\_M\_1511\_loop\_1 := var\_M\_1511\_loop\_2\_A + var\_M\_1511\_loop\_2\_B$

act4: $var\_M\_15111\_seq\_1 := var\_M\_15111\_seq\_2\_B$

act5: $selectedItem\_1 := selectedItem\_2\_B$

end

Event treat_failure_2_2 ⟨ordinary⟩ $\widehat{=}$

extends treat_failure_2

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = NOT\_OK$
grd4: $failureStatus\_1 = OK$
grd5: $var\_M\_15111\_seq\_2\_A \neq 0$
grd6: $var\_M\_15111\_seq\_2\_B \neq 0$

then

act1: $var\_M\_15\_cho := 1$
act2: $carts\_ref := \{p \cdot p \in \mathrm{ran}(carts\_ref) | site\_1 \mapsto p\}$
act3: $var\_M\_1511\_loop\_1 := var\_M\_1511\_loop\_2\_A + var\_M\_1511\_loop\_2\_B$

end

Event selectItemInItemList_1 ⟨convergent⟩ $\widehat{=}$

any

someProduct

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
grd4: $var\_M\_1511\_loop\_1 > 0$
grd5: $var\_M\_15111\_seq\_1 = 1$
grd6: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_15111\_seq\_1 := var\_M\_15111\_seq\_1 - 1$
act2: $selectedItem\_1 := \{someProduct\}$

end

Event addSelectedItemToCart_1 ⟨convergent⟩ $\widehat{=}$

refines selection_oneWebsite_loop

any

item used to access the element in selectedItem_1

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
grd4: $var\_M\_1511\_loop\_1 > 0$
grd5: $var\_M\_15111\_seq\_1 = 0$
grd6: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_1 = \{p\}$
grd7: $selectedItem\_1 = \{item\}$

with

someProduct: $selectedItem\_1 = \{someProduct\}$

then

act1: $var\_M\_1511\_loop\_1 := var\_M\_1511\_loop\_1 - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_1 \mapsto item\}$

end

Event selection_oneWebsite ⟨convergent⟩ $\widehat{=}$

extends selection_oneWebsite

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 1$
grd3: $failureStatus\_1 = OK$
grd4: $var\_M\_1511\_loop\_1 = 0$

then

act1: $var\_M\_15\_cho := 0$

end

Event selectItemInItemList_2_A ⟨convergent⟩ $\widehat{=}$
any
someProduct
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_A = 1$
grd5: $var\_M\_1511\_loop\_2\_A > 0$
grd6: $var\_M\_15111\_seq\_2\_A = 1$
grd7: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$
then
act1: $var\_M\_15111\_seq\_2\_A := var\_M\_15111\_seq\_2\_A - 1$
act2: $selectedItem\_2\_A := \{someProduct\}$
end
Event addSelectedItemToCart_2_A ⟨convergent⟩ $\widehat{=}$
refines selection_twoWebsites_A_loop
any
item used to access the element in selectedItem_2_A
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_A = 1$
grd5: $var\_M\_1511\_loop\_2\_A > 0$
grd6: $var\_M\_15111\_seq\_2\_A = 0$
grd7: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \wedge selectedItem\_2\_A = \{p\}$
grd8: $selectedItem\_2\_A = \{item\}$
with
someProduct: $selectedItem\_2\_A = \{someProduct\}$
then
act1: $var\_M\_1511\_loop\_2\_A := var\_M\_1511\_loop\_2\_A - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_2\_A \mapsto item\}$
end
Event selection_twoWebsites_A ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_A
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $var\_M\_151\_par\_A = 1$
grd4: $failureStatus\_2 = OK$
grd5: $var\_M\_1511\_loop\_2\_A = 0$
then
act1: $var\_M\_151\_par\_A := var\_M\_151\_par\_A - 1$
end
Event selectItemInItemList_2_B ⟨convergent⟩ $\widehat{=}$
any
someProduct
where
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_B = 1$
grd5: $var\_M\_1511\_loop\_2\_B > 0$

grd6: $var\_M\_15111\_seq\_2\_B = 1$
grd7: $someProduct \in P \setminus \mathrm{ran}(carts\_ref)$

then

act1: $var\_M\_15111\_seq\_2\_B := var\_M\_15111\_seq\_2\_B - 1$
act2: $selectedItem\_2\_B := \{someProduct\}$

end

Event addSelectedItemToCart_2_B ⟨convergent⟩ $\widehat{=}$
refines selection_twoWebsites_B_loop

any

item used to access the element in selectedItem_2_B

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_B = 1$
grd5: $var\_M\_1511\_loop\_2\_B > 0$
grd6: $var\_M\_15111\_seq\_2\_B = 0$
grd7: $\exists p \cdot p \in P \setminus \mathrm{ran}(carts\_ref) \land selectedItem\_2\_B = \{p\}$
grd8: $selectedItem\_2\_B = \{item\}$

with

someProduct: $selectedItem\_2\_B = \{someProduct\}$

then

act1: $var\_M\_1511\_loop\_2\_B := var\_M\_1511\_loop\_2\_B - 1$
act2: $carts\_ref := carts\_ref \cup \{site\_2\_B \mapsto item\}$

end

Event selection_twoWebsites_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_B

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $var\_M\_151\_par\_B = 1$
grd4: $failureStatus\_2 = OK$
grd5: $var\_M\_1511\_loop\_2\_B = 0$

then

act1: $var\_M\_151\_par\_B := var\_M\_151\_par\_B - 1$

end

Event selection_twoWebsites_join_A_B ⟨convergent⟩ $\widehat{=}$
extends selection_twoWebsites_join_A_B

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_15\_cho = 2$
grd3: $failureStatus\_2 = OK$
grd4: $var\_M\_151\_par\_A = 0$
grd5: $var\_M\_151\_par\_B = 0$
grd6: $var\_M\_1511\_loop\_2\_A = 0$
grd7: $var\_M\_1511\_loop\_2\_B = 0$

then

act1: $var\_M\_15\_cho := 0$

end

Event confirmSelection ⟨convergent⟩ $\widehat{=}$
extends confirmSelection

when

grd1: $var\_M\_1\_seq = 4$

grd3: $\mathrm{ran}(carts\_ref) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(carts\_ref) \Rightarrow \mathrm{card}(carts\_ref^{-1}[\{p\}]) = 1$
grd5: $var\_M\_15\_cho = 0$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := carts\_ref$
end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
when
grd1: $var\_M\_1\_seq = 3$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
when
grd1: $var\_M\_1\_seq = 2$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery
when
grd1: $var\_M\_1\_seq = 1$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
END

MACHINE M_16_failure_N
REFINES M_1_
SEES C_11_failure_status
VARIABLES

var_M_1_seq -
carts -
nb_sys -
var_M_16_cho number (id) of the current system that we are using
failureStatus

INVARIANTS

type1: $nb\_sys \in \mathbb{N}_1$
number of systems
type2: $var\_M\_16\_cho \in 0 \,..\, nb\_sys$
type3: $failureStatus \in 1 \,..\, nb\_sys \leftrightarrow FAILURE\_STATUS$

VARIANT

$var\_M\_1\_seq + var\_M\_16\_cho$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$
act3: $carts := \varnothing$
act4: $nb\_sys, failureStatus, var\_M\_16\_cho :|$
$nb\_sys' \in \mathbb{N}_1$
$\wedge\ failureStatus' = \{n \cdot n \in 1 \,..\, nb\_sys' | n \mapsto OK\}$
$\wedge\ var\_M\_16\_cho' \in 1 \,..\, nb\_sys'$

end

Event failure_n ⟨ordinary⟩ $\widehat{=}$

any

n

where

grd1: $var\_M\_1\_seq = 4$
grd2: $n \in \mathrm{dom}(failureStatus \rhd \{OK\})$

then

act1: $failureStatus := \{n \mapsto NOT\_OK\} \cup (\{n\} \lhd\!\!\!- failureStatus)$

end

Event treat_failure ⟨ordinary⟩ $\widehat{=}$

any

n

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{NOT\_OK\})$
the current system has failed
grd3: $n \in \mathrm{dom}(failureStatus \rhd \{OK\})$

then

act1: $var\_M\_16\_cho := n$

end

Event complete_failure ⟨ordinary⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$
grd2: $\mathrm{dom}(failureStatus \rhd \{OK\}) = \varnothing$

then

skip

end

Event selection_n ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{OK\})$

the current system is OK

then

act1: $var\_M\_16\_cho := 0$

end

Event selection ⟨convergent⟩ $\widehat{=}$

extends selection

any

$someCarts$

where

grd1: $var\_M\_1\_seq = 4$

grd2: $someCarts \subseteq SITES \times P$

grd3: $\mathrm{ran}(someCarts) = P$

grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$

grd5: $var\_M\_16\_cho = 0$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

act2: $carts := someCarts$

end

Event payment ⟨convergent⟩ $\widehat{=}$

extends payment

when

grd1: $var\_M\_1\_seq = 3$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event billing ⟨convergent⟩ $\widehat{=}$

extends billing

when

grd1: $var\_M\_1\_seq = 2$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event delivery ⟨convergent⟩ $\widehat{=}$

extends delivery

when

grd1: $var\_M\_1\_seq = 1$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

END

MACHINE M_161_
REFINES M_16_failure_N
SEES C_11_failure_status
VARIABLES

var_M_1_seq -

carts -

nb_sys -

var_M_16_cho number (id) of the current system that we are using

failureStatus

EVENTS

Initialisation

begin

act1: $var\_M\_1\_seq := 4$

act3: $carts := \varnothing$

act4: $nb\_sys, failureStatus, var\_M\_16\_cho :|$
$$nb\_sys' = 2 \wedge failureStatus' = \{n \cdot n \in 1 \,..\, nb\_sys' | n \mapsto OK\} \wedge var\_M\_16\_cho' \in 1 \,..\, nb\_sys'$$

end

Event failure_n ⟨ordinary⟩ $\widehat{=}$
extends failure_n

any

n

where

grd1: $var\_M\_1\_seq = 4$

grd2: $n \in \mathrm{dom}(failureStatus \rhd \{OK\})$

then

act1: $failureStatus := \{n \mapsto NOT\_OK\} \cup (\{n\} \lhd\!\!\!- failureStatus)$

end

Event treat_failure ⟨ordinary⟩ $\widehat{=}$
extends treat_failure

any

n

where

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{NOT\_OK\})$
the current system has failed

grd3: $n \in \mathrm{dom}(failureStatus \rhd \{OK\})$

then

act1: $var\_M\_16\_cho := n$

end

Event complete_failure ⟨ordinary⟩ $\widehat{=}$
extends complete_failure

when

grd1: $var\_M\_1\_seq = 4$

grd2: $\mathrm{dom}(failureStatus \rhd \{OK\}) = \varnothing$

then

skip

end

Event selection_sys1 ⟨convergent⟩ $\widehat{=}$
extends selection_n

when

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{OK\})$
the current system is OK
grd3: $var\_M\_16\_cho = 1$
then
act1: $var\_M\_16\_cho := 0$
end
Event selection_sys2 ⟨convergent⟩ $\widehat{=}$
extends selection_n
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{OK\})$
the current system is OK
grd3: $var\_M\_16\_cho = 2$
then
act1: $var\_M\_16\_cho := 0$
end
Event selection ⟨convergent⟩ $\widehat{=}$
extends selection
any
$someCarts$
where
grd1: $var\_M\_1\_seq = 4$
grd2: $someCarts \subseteq SITES \times P$
grd3: $\mathrm{ran}(someCarts) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$
grd5: $var\_M\_16\_cho = 0$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := someCarts$
end
Event payment ⟨convergent⟩ $\widehat{=}$
extends payment
when
grd1: $var\_M\_1\_seq = 3$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event billing ⟨convergent⟩ $\widehat{=}$
extends billing
when
grd1: $var\_M\_1\_seq = 2$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
Event delivery ⟨convergent⟩ $\widehat{=}$
extends delivery
when
grd1: $var\_M\_1\_seq = 1$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
end
END

MACHINE M_1611_
REFINES M_161_
SEES C_11_failure_status
VARIABLES

var_M_1_seq -
carts -
nb_sys -
var_M_16_cho number (id) of the current system that we are using
failureStatus -
var_M_1611_par_A -
var_M_1611_par_B -

INVARIANTS

type1: $var\_M\_1611\_par\_A \in \mathbb{N}$
type2: $var\_M\_1611\_par\_B \in \mathbb{N}$

VARIANT

$var\_M\_1\_seq + var\_M\_16\_cho + var\_M\_1611\_par\_A + var\_M\_1611\_par\_B$

EVENTS

Initialisation ⟨extended⟩

begin

act1: $var\_M\_1\_seq := 4$
act3: $carts := \varnothing$
act4: $nb\_sys, failureStatus, var\_M\_16\_cho :\mid$
$nb\_sys' = 2$
$\land\ failureStatus' = \{n \cdot n \in 1 \mathinner{..} nb\_sys' | n \mapsto OK\}$
$\land\ var\_M\_16\_cho' \in 1 \mathinner{..} nb\_sys'$
act5: $var\_M\_1611\_par\_A := 1$
act6: $var\_M\_1611\_par\_B := 1$

end

Event failure_n ⟨ordinary⟩ $\widehat{=}$
extends failure_n

any

n

where

grd1: $var\_M\_1\_seq = 4$
grd2: $n \in \operatorname{dom}(failureStatus \rhd \{OK\})$

then

act1: $failureStatus := \{n \mapsto NOT\_OK\} \cup (\{n\} \lhd\!\!\!- failureStatus)$

end

Event treat_failure ⟨ordinary⟩ $\widehat{=}$
extends treat_failure

any

n

where

grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_16\_cho \in \operatorname{dom}(failureStatus \rhd \{NOT\_OK\})$
the current system has failed
grd3: $n \in \operatorname{dom}(failureStatus \rhd \{OK\})$

then

act1: $var\_M\_16\_cho := n$

end

Event complete_failure ⟨ordinary⟩ $\widehat{=}$

extends complete_failure
when
grd1: $var\_M\_1\_seq = 4$
grd2: $\mathrm{dom}(failureStatus \rhd \{OK\}) = \varnothing$
then
skip
end
Event selection_sys1 ⟨convergent⟩ $\widehat{=}$
extends selection_sys1
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{OK\})$
the current system is OK
grd3: $var\_M\_16\_cho = 1$
then
act1: $var\_M\_16\_cho := 0$
end
Event selection_sys2_B ⟨convergent⟩ $\widehat{=}$
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_16\_cho = 2$
grd3: $var\_M\_1611\_par\_B = 1$
grd4: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{OK\})$
then
act1: $var\_M\_1611\_par\_B := var\_M\_1611\_par\_B - 1$
end
Event selection_sys2_join_AB ⟨convergent⟩ $\widehat{=}$
extends selection_sys2
when
grd1: $var\_M\_1\_seq = 4$
grd2: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{OK\})$
the current system is OK
grd3: $var\_M\_16\_cho = 2$
grd4: $var\_M\_1611\_par\_A = 0$
grd5: $var\_M\_1611\_par\_B = 0$
then
act1: $var\_M\_16\_cho := 0$
end
Event selection ⟨convergent⟩ $\widehat{=}$
extends selection
any
someCarts
where
grd1: $var\_M\_1\_seq = 4$
grd2: $someCarts \subseteq SITES \times P$
grd3: $\mathrm{ran}(someCarts) = P$
grd4: $\forall p \cdot p \in \mathrm{ran}(someCarts) \Rightarrow \mathrm{card}(someCarts^{-1}[\{p\}]) = 1$
grd5: $var\_M\_16\_cho = 0$
then
act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$
act2: $carts := someCarts$
end
Event payment ⟨convergent⟩ $\widehat{=}$

extends payment

when

grd1: $var\_M\_1\_seq = 3$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event billing ⟨convergent⟩ $\widehat{=}$

extends billing

when

grd1: $var\_M\_1\_seq = 2$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event delivery ⟨convergent⟩ $\widehat{=}$

extends delivery

when

grd1: $var\_M\_1\_seq = 1$

then

act1: $var\_M\_1\_seq := var\_M\_1\_seq - 1$

end

Event selection_sys2_A ⟨convergent⟩ $\widehat{=}$

when

grd1: $var\_M\_1\_seq = 4$

grd2: $var\_M\_16\_cho = 2$

grd3: $var\_M\_1611\_par\_A = 1$

grd4: $var\_M\_16\_cho \in \mathrm{dom}(failureStatus \rhd \{OK\})$

then

act1: $var\_M\_1611\_par\_A := var\_M\_1611\_par\_A - 1$

end

END

C Hybrid systems: Continuous to discrete models

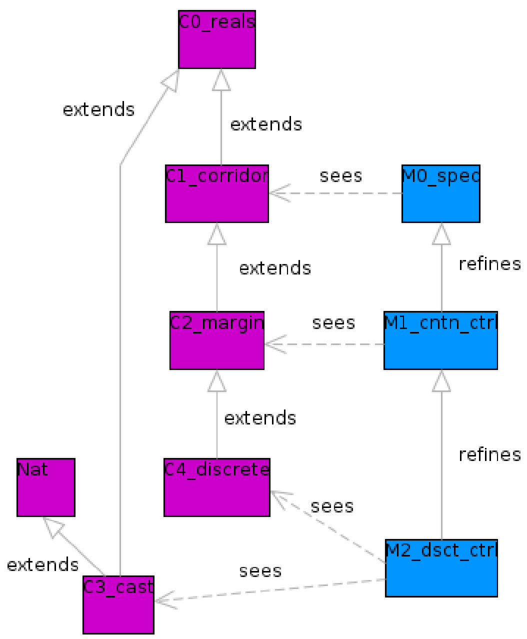
For technical reasons, the names in the actual complete models are slightly different from those in the partial models of Chapter 6 (which are more consistent):

	Chapter 6	Rodin models
M0	fv	p
	new_fv	new_p
M1	fc	pc
	nfc	np
M2	fd	pd

Components:

- *C0_reals* theorems about functions and reals (page 221)
- *C1_corridor* definition of the safety envelope (page 224)
- *M0_spec* abstract controller (page 225)
- *C2_margin* definition of the safety margin (page 227)
- *M1_cntn_ctrl* continuous controller (page 228)
- *Nat* induction on naturals (page 232)
- *C3_cast* (page 233)
- *C4_discrete* definition of the discrete time step (page 235)
- *M2_dsct_ctrl* discrete controller (page 236)

Theories used in this development: *Real* (page 140) and *RealPos* (page 145)
The models are also available at: `http://babin.perso.enseeiht.fr/r/thesis/`


C0_reals
extends
extends
C1_corridor
sees
M0_spec
extends
refines
C2_margin
sees
M1_cntn_ctrl
extends
refines
Nat
C4_discrete
sees
extends
sees
M2_dsct_ctrl
C3_cast

CONTEXT C0_reals

theorems concerning continuous mathematical functions

CONSTANTS

REAL_POS

REAL_STR_POS

AXIOMS

def01: $REAL\_POS = \{x | x \in REAL \land \mathrm{leq}(zero, x)\}$

def02: $REAL\_STR\_POS = \{x | x \in REAL \land \mathrm{smr}(zero, x)\}$

thm01: ⟨theorem⟩ $REAL\_POS \subseteq REAL$

thm02: ⟨theorem⟩ $REAL\_STR\_POS \subseteq REAL\_POS$

thm03: ⟨theorem⟩ $REAL\_STR\_POS \subseteq REAL$

thm39: ⟨theorem⟩ $\forall a, b \cdot a \in REAL \land b \in REAL \Rightarrow (a = a \,\mathrm{plus}\, b \Rightarrow b = zero)$

thm04: ⟨theorem⟩ $zero \in REAL\_POS$

thm05: ⟨theorem⟩ $\mathrm{leq}(zero, zero)$

thm06: ⟨theorem⟩ $\forall n, A, f, a \cdot n \in \mathbb{N}$
$\land A \subseteq REAL$
$\land f \in 0 \,..\, n \to A$
$\land a \in A$
$\Rightarrow f \cup \{n + 1 \mapsto a\} \in 0 \,..\, n + 1 \to A$

thm07: ⟨theorem⟩ $\forall a, b, c \cdot (a \in REAL \land b \in REAL \land c \in REAL) \Rightarrow$
$(\mathrm{leq}(a \,\mathrm{plus}\, c, b \,\mathrm{plus}\, c) \Leftrightarrow \mathrm{leq}(a, b))$

a+c ≤ b+c ⇔ a≤ b

thm08: ⟨theorem⟩ $\forall x \cdot x \in REAL \Rightarrow$
$(\mathrm{leq}(zero, x) \Leftrightarrow \mathrm{leq}(\mathrm{minus}(x), zero))$

0≤ x ⇔ -x≤ 0

thm09: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(a, b) \Leftrightarrow \mathrm{leq}(zero, b \,\mathrm{sub}\, a))$

a≤ b ⇔ 0≤ b-a

thm10: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(zero, a) \Leftrightarrow \mathrm{leq}(b, b \,\mathrm{plus}\, a))$

0≤ a ⇔ b≤ b+a

thm11: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(zero, b) \Rightarrow \mathrm{leq}(a, a \,\mathrm{plus}\, b))$

0≤ b ⇒ a ≤ a+b

thm14: ⟨theorem⟩ $\forall a, b \cdot a \in REAL \land b \in REAL \Rightarrow$
$(a = b \Leftrightarrow b = a)$

a=b ⇔ b=a

thm13: ⟨theorem⟩ $\forall a, b \cdot a \in REAL \land b \in REAL \Rightarrow$
$(\neg(a = b) \Leftrightarrow \neg(b = a))$

¬(a=b) ⇔ ¬(b=a)

thm12: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{smr}(zero, b) \Rightarrow \mathrm{smr}(a, a \,\mathrm{plus}\, b))$

0<b ⇒ a < a+b

thm33: ⟨theorem⟩ $\forall a \cdot zero \,\mathrm{mult}\, a = zero$

0 * a = 0

thm38: ⟨theorem⟩ $\forall a \cdot a \,\mathrm{mult}\, \mathrm{minus}(one) = \mathrm{minus}(a)$

a*(−1) = −a

thm41: ⟨theorem⟩ $\forall a \cdot \mathrm{minus}(\mathrm{minus}(a)) = a$

−(−a) = a

thm17: ⟨theorem⟩ $\mathrm{leq}(zero, one)$

0≤ 1

thm15: ⟨theorem⟩ $\mathrm{smr}(zero, one)$
0<1

thm34: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(zero, b) \Rightarrow \mathrm{leq}(a \,\mathrm{sub}\, b, a))$
0≤ b ⇒ a-b ≤ a

thm16: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{smr}(zero, b) \Rightarrow \mathrm{smr}(a \,\mathrm{sub}\, b, a))$
0<b ⇒ a-b < a

thm18: $\forall a, b, c, f \cdot (a \in REAL \land b \in REAL \land c \in REAL \land \mathrm{leq}(a, b) \land \mathrm{leq}(b, c)$
$\land f \in REAL \nrightarrow REAL \land \{x | x \in REAL \land \mathrm{leq}(a, x) \land \mathrm{leq}(x, c)\} \subseteq \mathrm{dom}(f)) \Rightarrow$
$(\mathrm{cnt\_int}(f, a, c) \Leftrightarrow \mathrm{cnt\_int}(f, a, b) \land \mathrm{cnt\_int}(f, b, c))$
continuous on [a,c] ⇔ continuous on [a,b] and [b,c]

thm19: $\forall a, b, f, g \cdot (a \in REAL \land b \in REAL \land \mathrm{leq}(a, b)$
$\land f \in REAL \nrightarrow REAL \land \{x | x \in REAL \land \mathrm{leq}(a, x) \land \mathrm{leq}(x, b)\} \subseteq \mathrm{dom}(f)$
$\land g \in REAL \nrightarrow REAL \land \{x | x \in REAL \land \mathrm{leq}(a, x) \land \mathrm{leq}(x, b)\} \subseteq \mathrm{dom}(g)$
$\land (\forall x \cdot x \in REAL \land \mathrm{leq}(a, x) \land \mathrm{leq}(x, b) \Rightarrow f(x) = g(x))) \Rightarrow$
$(\mathrm{cnt\_int}(f, a, b) \Leftrightarrow \mathrm{cnt\_int}(g, a, b))$
f and g equal on [a,b] ⇒ (f continuous on [a,b] ⇔ g continuous on [a,b])

thm20: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(a, b) \land \mathrm{leq}(b, a) \Leftrightarrow a = b)$
a≤ b ∧ b≤ a ⇔ a=b

thm21: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\neg \mathrm{leq}(a, b) \Leftrightarrow \mathrm{gtr}(a, b))$
¬(a≤ b) ⇔ a>b

thm22: $\forall a, b \cdot (a \in REAL\_POS \land b \in REAL\_POS) \Rightarrow$
$(a \,\mathrm{mult}\, b \in REAL\_POS)$
a ∈ R+ ∧ b ∈ R+ ⇒ a*b ∈ R+

thm23: $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$((\exists c \cdot c \in REAL\_STR\_POS \land a = b \,\mathrm{plus}\, c) \Leftrightarrow \mathrm{smr}(b, a))$
(∃ c > 0, a = b+c) ⇔ b<a

thm24: $\forall a, b, c \cdot (a \in REAL \land b \in REAL \land c \in REAL) \Rightarrow$
$(\mathrm{smr}(a, b) \land \mathrm{smr}(b, c) \Rightarrow \mathrm{smr}(a, c))$
a<b ∧ b<c ⇒ a<c

thm26: ⟨theorem⟩ $\forall a, b, c \cdot (a \in REAL \land b \in REAL \land c \in REAL) \Rightarrow$
$(\mathrm{leq}(a, b) \land \mathrm{smr}(b, c) \Rightarrow \mathrm{smr}(a, c))$
a≤ b ∧ b<c ⇒ a<c

thm25: $\forall a, b, now \cdot now \in REAL\_POS \land a \in REAL\_POS \land b \in REAL\_POS \land \mathrm{smr}(a, b) \Rightarrow$
$(\exists dt, np \cdot$
$dt \in REAL\_STR\_POS \land$
$np \in REAL\_POS \nrightarrow REAL\_POS \land$
$\mathrm{dom}(np) = \{t \cdot \mathrm{leq}(now, t) \land \mathrm{leq}(t, now \,\mathrm{plus}\, dt) | t\} \land$
$np(now) = a \land$
$np(now \,\mathrm{plus}\, dt) = b \land$
$(\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \land t2 \in \mathrm{dom}(np) \land \mathrm{smr}(t1, t2) \Rightarrow \mathrm{smr}(np(t1), np(t2))) \land$

$\mathrm{cnt\_int}(np, now, now \,\mathrm{plus}\, dt))$
∀a,b∈ R+, there exists a continuous and strictly increasing function on [now,now+dt] whose range is [a,b]

thm28: $\forall a, b, now \cdot now \in REAL\_POS \land a \in REAL\_POS \land b \in REAL\_POS \land \mathrm{leq}(a, b) \Rightarrow$
$(\exists dt, np \cdot$
$dt \in REAL\_STR\_POS \land$
$np \in REAL\_POS \nrightarrow REAL\_POS \land$

$\mathrm{dom}(np) = \{t \cdot \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt) | t\} \wedge$
$np(now) = a \wedge$
$np(now \text{ plus } dt) = b \wedge$
$(\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t1), np(t2))) \wedge$
$\mathrm{cnt\_int}(np, now, now \text{ plus } dt))$

∀a,b∈ R+, there exists a continuous and increasing function on [now,now+dt] whose range is [a,b]

thm29: $\forall a, b, now \cdot now \in REAL\_POS \wedge a \in REAL\_POS \wedge b \in REAL\_POS \wedge \mathrm{leq}(b, a) \Rightarrow$
$(\exists dt, np \cdot$
$dt \in REAL\_STR\_POS \wedge$
$np \in REAL\_POS \nrightarrow REAL\_POS \wedge$
$\mathrm{dom}(np) = \{t \cdot \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt) | t\} \wedge$
$np(now) = a \wedge$
$np(now \text{ plus } dt) = b \wedge$
$(\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t2), np(t1))) \wedge$
$\mathrm{cnt\_int}(np, now, now \text{ plus } dt))$

∀a,b∈ R+, there exists a continuous and decreasing function on [now,now+dt] whose range is [a,b]

thm27: ⟨theorem⟩ $\forall a, b \cdot \mathrm{leq}(a, b) \vee \mathrm{leq}(b, a)$

a≤ b ∨ b≤ a

thm30: $\forall a, b, c \cdot (a \in REAL\_POS \wedge b \in REAL\_POS \wedge c \in REAL\_STR\_POS) \Rightarrow$
$(\mathrm{smr}(a, b) \Rightarrow \mathrm{smr}(a \text{ mult } c, b \text{ mult } c))$

a ≥ 0 ∧ b ≥ 0 ∧ c > 0 ⇒ (a < b ⇒ a*c < b*c)

thm31: $\forall a, b, c \cdot (a \in REAL\_POS \wedge b \in REAL\_POS \wedge c \in REAL\_POS) \Rightarrow$
$(\mathrm{leq}(a, b) \Rightarrow \mathrm{leq}(a \text{ mult } c, b \text{ mult } c))$

a ≥ 0 ∧ b ≥ 0 ∧ c ≥ 0 ⇒ (a ≤ b ⇒ a*c ≤ b*c)

thm40: $\forall a, b, c \cdot (a \in REAL\_POS \wedge b \in REAL\_POS \wedge c \in REAL\_STR\_POS) \Rightarrow$
$(\mathrm{leq}(a \text{ mult } c, b \text{ mult } c) \Rightarrow \mathrm{leq}(a, b))$

a ≥ 0 ∧ b ≥ 0 ∧ c > 0 ⇒ (a*c ≤ b*c ⇒ a ≤ b)

thm32: ⟨theorem⟩ $\forall a, b \cdot \mathrm{smr}(a, b) \Leftrightarrow \neg \mathrm{leq}(b, a)$

a<b ⇔ ¬ b≤ a

thm35: $\forall a \cdot a \in REAL\_STR\_POS \Rightarrow ($
$\exists b \cdot b \in REAL\_STR\_POS \wedge \mathrm{smr}(b, a))$

∀ a > 0, ∃ b > 0, b < a

thm36: $\forall a, b \cdot \mathrm{smr}(a, b) \Leftrightarrow \mathrm{smr}(zero, b \text{ sub } a)$

a < b ⇔ 0 < b − a

thm37: $\forall a, b, c \cdot \mathrm{smr}(a, b) \Leftrightarrow \mathrm{smr}(a \text{ plus } c, b \text{ plus } c)$

a < b ⇔ a+c < b+c

END

CONTEXT C1_corridor
energy corridor
EXTENDS C0_reals
CONSTANTS
m
M
AXIOMS
axm01: $m \in REAL\_STR\_POS$
axm02: $M \in REAL\_STR\_POS$
axm03: $\mathrm{smr}(m, M)$
thm01: ⟨theorem⟩ $\mathrm{leq}(m, M)$
thm02: ⟨theorem⟩ $\mathrm{leq}(zero, m)$
thm06: ⟨theorem⟩ $\mathrm{leq}(zero, M)$
thm03: ⟨theorem⟩ $\forall x \cdot \mathrm{leq}(m, x) \Rightarrow x \in REAL\_POS$
thm04: ⟨theorem⟩ $\mathrm{leq}(m, m)$
thm05: ⟨theorem⟩ $\forall a \cdot \mathrm{leq}(m, a) \Rightarrow \mathrm{leq}(zero, a)$
END

MACHINE M0_spec
SEES C1_corridor
VARIABLES

p
active

INVARIANTS

inv01: $p \in REAL\_POS$
inv02: $active \in$ BOOL
inv03: $active =$ TRUE $\Rightarrow$ leq$(m, p) \wedge$ leq(p, M)
active ⇒ p ∈ [m,M]
inv04: $active =$ FALSE $\Rightarrow p = zero$
¬active ⇒ p = 0
thm01: ⟨theorem⟩ leq$(zero, p) \wedge$ leq(p, M)
p ∈ [0,M]
DLF: ⟨theorem⟩ (
$(active =$ FALSE$)$
$\wedge\ (p = zero)$
$) \vee ($
$\exists new\_p \cdot ($
$(active =$ TRUE$)$
$\wedge\ (new\_p \in REAL\_POS)$
$\wedge$ (leq$(m, new_p) \wedge$ leq$(new_p, M))$
)
$) \vee ($
$(active =$ TRUE$)$
$\wedge$ (leq$(m, p) \wedge$ leq$(p, M))$
)
at least one event is enabled
deterministic1: ⟨theorem⟩ ¬(
(
$(active =$ FALSE$)$
$\wedge\ (p = zero)$
$) \wedge ($
$\exists new\_p \cdot ($
$(active =$ TRUE$)$
$\wedge\ (new\_p \in REAL\_POS)$
$\wedge$ (leq$(m, new_p) \wedge$ leq$(new_p, M))$
)
)
)
events 'start' and 'produce' are never enabled simultaneously
deterministic2: ⟨theorem⟩ ¬(
(
$(active =$ FALSE$)$
$\wedge\ (p = zero)$
$) \wedge ($
$(active =$ TRUE$)$
$\wedge$ (leq$(m, p) \wedge$ leq$(p, M))$
)
)
events 'start' and 'stop' are never enabled simultaneously

EVENTS
Initialisation

begin
act02: $active := \text{FALSE}$
act01: $p := zero$
end
Event start ⟨ordinary⟩ $\widehat{=}$
when
grd02: $active = \text{FALSE}$
grd01: $p = zero$
then
act01: $active := \text{TRUE}$
act02: $p :\mid \text{leq}(m, p') \wedge \text{leq}(p', M)$
end
Event produce ⟨ordinary⟩ $\widehat{=}$
any
new_p
where
grd02: $active = \text{TRUE}$
grd03: $new\_p \in REAL\_POS$
grd01: $\text{leq}(m, new\_p) \wedge \text{leq}(new\_p, M)$
then
act01: $p := new\_p$
end
Event stop ⟨ordinary⟩ $\widehat{=}$
when
grd02: $active = \text{TRUE}$
grd01: $\text{leq}(m, p) \wedge \text{leq}(p, M)$
then
act01: $active := \text{FALSE}$
act02: $p := zero$
end
END

CONTEXT C2_margin

energy corridor margin

EXTENDS C1_corridor

CONSTANTS

z

AXIOMS

axm01: $z \in REAL\_POS$

z ∈ R+

axm02: $\mathrm{gtr}(M \mathrm{\ sub\ } m, (one \mathrm{\ plus\ } one) \mathrm{\ mult\ } z)$

M−m > 2*z

thm01: ⟨theorem⟩ $\mathrm{leq}(zero, z)$

0 ≤ z

thm02: ⟨theorem⟩ $\mathrm{leq}(zero, m \mathrm{\ plus\ } z)$

0 ≤ m+z

thm09: ⟨theorem⟩ $\mathrm{leq}(z, M)$

z ≤ M

thm03: ⟨theorem⟩ $\mathrm{leq}(zero, M \mathrm{\ sub\ } z)$

0 ≤ M−z

thm04: ⟨theorem⟩ $\mathrm{leq}(m, m \mathrm{\ plus\ } z)$

m ≤ m+z

thm05: ⟨theorem⟩ $\mathrm{leq}(M \mathrm{\ sub\ } z, M)$

M−z ≤ M

thm06: ⟨theorem⟩ $\mathrm{leq}(z, M \mathrm{\ sub\ } m)$

z ≤ M−m

thm07: ⟨theorem⟩ $\mathrm{leq}(m, M \mathrm{\ sub\ } z)$

m ≤ M−z

thm08: ⟨theorem⟩ $\mathrm{leq}(m \mathrm{\ plus\ } z, M)$

m+z ≤ M

thm10: ⟨theorem⟩ $\mathrm{leq}(m \mathrm{\ plus\ } z, M \mathrm{\ sub\ } z)$

m+z ≤ M−z

END

MACHINE M1_cntn_ctrl
REFINES M0_spec
SEES C2_margin
VARIABLES

p
active
now
pc
active_t has a sense only if active is TRUE ; time (moment) when S became active

INVARIANTS

type01: $now \in REAL\_POS$
type02: $pc \in REAL\_POS \rightarrow REAL\_POS$
type03: $active\_t \in REAL\_POS$
glue01: $p = pc(now)$
prop01: $\mathrm{cnt\_int}(pc, zero, now)$
pc is continous on [0,now]
prop02: $active = \mathrm{TRUE} \Rightarrow$
$(\forall t \cdot t \in REAL \wedge \mathrm{leq}(active\_t, t) \wedge \mathrm{leq}(t, now) \Rightarrow$
$(\mathrm{leq}(m \text{ plus } z, pc(t)) \wedge \mathrm{leq}(pc(t), M \text{ sub } z)))$
(x = S ∧ active) ⇒ (∀ t ∈ [active_t,now], pc(t) ∈ [m+z,M−z])
prop03: $\forall t \cdot t \in REAL \wedge \mathrm{leq}(zero, t) \wedge \mathrm{leq}(t, now) \Rightarrow \mathrm{leq}(pc(t), M)$
∀ t ∈ [0,now] ⇒ pc(t) ≤ M
prop04: $active = \mathrm{TRUE} \Rightarrow \mathrm{leq}(active\_t, now)$
DLF__start__produce: ⟨theorem⟩ (
$\exists dt, np \cdot ($
$(active = \mathrm{FALSE})$
$\wedge (p = zero)$
$\wedge (dt \in REAL\_STR\_POS)$
$\wedge (np \in REAL\_POS \nrightarrow REAL\_POS)$
$\wedge (\mathrm{dom}(np) = \{t \cdot t \in REAL \wedge \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt) | t\})$
$\wedge (np(now) = pc(now))$
$\wedge (np(now \text{ plus } dt) = m \text{ plus } z)$
$\wedge (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{smr}(t1, t2) \Rightarrow \mathrm{smr}(np(t1), np(t2)))$
$\wedge (\mathrm{cnt\_int}(np, now, now \text{ plus } dt))$
)
) ∨ (
$\exists new\_p, dt, np \cdot ($
$(active = \mathrm{TRUE})$
$\wedge (new\_p \in REAL\_POS)$
$\wedge (\mathrm{leq}(m, new\_p) \wedge \mathrm{leq}(new\_p, M))$
$\wedge (dt \in REAL\_STR\_POS)$
$\wedge (np \in REAL\_POS \nrightarrow REAL\_POS)$
$\wedge (\mathrm{dom}(np) = \{t \cdot t \in REAL \wedge \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt) | t\})$
$\wedge (np(now) = pc(now))$
$\wedge (np(now \text{ plus } dt) = new\_p)$
$\wedge (\mathrm{leq}(p, new\_p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t1), np(t2))))$
$\wedge (\mathrm{leq}(new\_p, p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t2), np(t1))))$
$\wedge (\mathrm{cnt\_int}(np, now, now \text{ plus } dt))$
$\wedge (\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(m \text{ plus } z, np(t)) \wedge \mathrm{leq}(np(t), M \text{ sub } z))$
)
)

EVENTS

Initialisation ⟨extended⟩

begin

act02: $active := \text{FALSE}$

act01: $p := zero$

act06: $now := zero$

act07: $pc := \lambda t \cdot t \in REAL\_POS | zero$

act08: $active\_t :\in REAL\_POS$

end

Event start ⟨ordinary⟩ $\widehat{=}$

refines start

any

dt

np

where

grd02: $active = \text{FALSE}$

grd01: $p = zero$

grd04: $dt \in REAL\_STR\_POS$

dt > 0

grd05: $np \in REAL\_POS \twoheadrightarrow REAL\_POS$

np ∈ R+ ⇸ R+

grd06: $\text{dom}(np) = \{t | t \in REAL \wedge \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt)\}$

dom(np) = [now,now+dt]

grd07: $np(now) = pc(now)$

np(now) = pc(now)

grd08: $np(now \text{ plus } dt) = m \text{ plus } z$

np(now+dt) = m+z

grd09: $\forall t1, t2 \cdot t1 \in \text{dom}(np) \wedge t2 \in \text{dom}(np) \wedge \text{smr}(t1, t2) \Rightarrow$
$\text{smr}(np(t1), np(t2))$

np is a monotonically strictly increasing function : a<b ⇒ np(a)<np(b)

grd12: $\text{cnt\_int}(np, now, now \text{ plus } dt)$

np is continuous on [now,now+dt]

thm02: ⟨theorem⟩ $\text{smr}(now, now \text{ plus } dt)$

thm01: ⟨theorem⟩ $\text{dom}(pc \Leftarrow np) = REAL\_POS$

then

act01: $active := \text{TRUE}$

act02: $p := m \text{ plus } z$

act03: $now := now \text{ plus } dt$

act04: $pc := pc \Leftarrow np$

act05: $active\_t := now \text{ plus } dt$

end

Event produce_safe ⟨ordinary⟩ $\widehat{=}$

extends produce

any

new_p

dt

np

where

grd02: $active = \text{TRUE}$

grd03: $new\_p \in REAL\_POS$

grd01: $\text{leq}(m, new\_p) \wedge \text{leq}(new\_p, M)$

grd10: $dt \in REAL\_STR\_POS$

dt > 0

thm02: ⟨theorem⟩ $\text{smr}(now, now \text{ plus } dt)$

grd11: $np \in REAL\_POS \nrightarrow REAL\_POS$
np ∈ R+ ⇸ R+
grd06: $\mathrm{dom}(np) = \{t | t \in REAL \wedge \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt)\}$
dom(np) = [now,now+dt]
grd07: $np(now) = pc(now)$
np(now) = pc(now)
grd08: $np(now \text{ plus } dt) = new\_p$
np(now+dt) = new_p
grd09: $\mathrm{leq}(p, new\_p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t1), np(t2)))$
np is a monotonic function
grd14: $\mathrm{leq}(new\_p, p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t2), np(t1)))$
np is a monotonic function
grd12: $\mathrm{cnt\_int}(np, now, now \text{ plus } dt)$
np is continuous on [now,now+dt]
thm01: ⟨theorem⟩ $\mathrm{dom}(pc \Leftarrow np) = REAL\_POS$
grd13: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(m \text{ plus } z, np(t)) \wedge \mathrm{leq}(np(t), M \text{ sub } z)$
∀ t ∈ [now,now+dt] ⇒ np(t) ∈ [m+z,M−z]

then

act01: $p := new\_p$
act02: $now := now \text{ plus } dt$
act03: $pc := pc \Leftarrow np$

end

Event safety_stop ⟨ordinary⟩ $\widehat{=}$

extends stop

any

dt
np

where

grd02: $active = \mathrm{TRUE}$
grd01: $\mathrm{leq}(m, p) \wedge \mathrm{leq}(p, M)$
grd10: $dt \in REAL\_STR\_POS$
dt > 0
thm02: ⟨theorem⟩ $\mathrm{smr}(now, now \text{ plus } dt)$
grd11: $np \in REAL\_POS \nrightarrow REAL\_POS$
np ∈ R+ ⇸ R+
grd06: $\mathrm{dom}(np) = \{t | t \in REAL \wedge \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt)\}$
dom(np) = [now,now+dt]
grd07: $np(now) = pc(now)$
np(now) = pc(now)
grd08: $np(now \text{ plus } dt) = zero$
np(now+dt) = 0
grd12: $\mathrm{cnt\_int}(np, now, now \text{ plus } dt)$
np is continuous on [now,now+dt]
thm01: ⟨theorem⟩ $\mathrm{dom}(pc \Leftarrow np) = REAL\_POS$
grd53: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(np(t), M)$
∀ t ∈ [now,now+dt] ⇒ np(t) ≤ M
grd54: $\exists t \cdot t \in \mathrm{dom}(np) \Rightarrow \neg(\mathrm{leq}(m \text{ plus } z, np(t)) \wedge \mathrm{leq}(np(t), M \text{ sub } z))$
∃ t ∈ [now,now+dt] ⇒ ¬ np(t) ∈ [m+z,M−z] ; safety risk

then

act01: $active := \mathrm{FALSE}$
act02: $p := zero$
act03: $now := now \text{ plus } dt$
act04: $pc := pc \Leftarrow np$

end

Event stop ⟨ordinary⟩ $\widehat{=}$

extends stop

any

np

dt

where

grd02: $active = \mathrm{TRUE}$

grd01: $\mathrm{leq}(m, p) \wedge \mathrm{leq}(p, M)$

grd04: $dt \in REAL\_STR\_POS$

dt > 0

thm02: ⟨theorem⟩ $\mathrm{smr}(now, now \text{ plus } dt)$

grd05: $np \in REAL\_POS \nrightarrow REAL\_POS$

np ∈ R+ ⇸ R+

grd06: $\mathrm{dom}(np) = \{t | t \in REAL \wedge \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt)\}$

dom(np) = [now,now+dt]

grd07: $np(now) = pc(now)$

np(now) = pc(now)

grd08: $np(now \text{ plus } dt) = zero$

np(now+dt) = 0

grd09: $\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{smr}(t1, t2) \Rightarrow \mathrm{gtr}(np(t1), np(t2))$

np is a monotonically strictly decreasing function : a<b ⇒ np(a)>np(b)

grd12: $\mathrm{cnt\_int}(np, now, now \text{ plus } dt)$

np is continuous on [now,now+dt]

thm01: ⟨theorem⟩ $\mathrm{dom}(pc \Leftarrow np) = REAL\_POS$

then

act01: $active := \mathrm{FALSE}$

act02: $p := zero$

act03: $now := now \text{ plus } dt$

act04: $pc := pc \Leftarrow np$

end

END

CONTEXT Nat

From: Hoang, Thai Son - 2013-01-21 09:53:01
Rodin-b-sharp-user Mailing List
http://sourceforge.net/p/rodin-b-sharp/mailman/message/30378566/

AXIOMS

`well-order`: ⟨theorem⟩ $\forall S \cdot S \subseteq \mathbb{N} \land S \neq \varnothing \Rightarrow (\exists m \cdot m \in S \land (\forall x \cdot x \in S \Rightarrow m \leq x))$

`induction`: ⟨theorem⟩ $\forall S \cdot S \subseteq \mathbb{N} \land 0 \in S \land (\forall x \cdot x \in S \Rightarrow x + 1 \in S) \Rightarrow \mathbb{N} \subseteq S$

END

CONTEXT C3_cast
EXTENDS C0_reals,Nat
CONSTANTS

cast

AXIOMS

axm01: $cast \in \mathbb{N} \to REAL\_POS$
type and domain

axm02: $cast(0) = zero$
initial case

axm03: $\forall a \cdot a \in \mathbb{N} \Rightarrow ($
$cast(a+1) = cast(a) \text{ plus } one)$
induction case

thm00: ⟨theorem⟩ $\mathrm{dom}(cast) = \mathbb{N}$

thm02: ⟨theorem⟩ $\mathrm{ran}(cast) = cast[\mathbb{N}]$

thm01: ⟨theorem⟩ $cast(1) = one$

thm04: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$cast(a+b) = cast(a) \text{ plus } cast(b))$
(proof by induction on b)

thm06: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$a < b \Rightarrow \mathrm{smr}(cast(a), cast(b)))$
(proof by induction on b)

thm07: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$a = b \Rightarrow cast(a) = cast(b))$

thm08: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$a \neq b \Rightarrow cast(a) \neq cast(b))$

thm09: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$a \leq b \Rightarrow \mathrm{leq}(cast(a), cast(b)))$

thm10: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$\mathrm{smr}(cast(a), cast(b)) \Rightarrow a < b)$

thm11: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$a < b \Leftrightarrow \mathrm{smr}(cast(a), cast(b)))$
equivalence over '<'

thm12: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$a = b \Leftrightarrow cast(a) = cast(b))$
equivalence over '='

thm13: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$a \neq b \Leftrightarrow cast(a) \neq cast(b))$
equivalence over '≠ '

thm14: ⟨theorem⟩ $\forall a, b \cdot (a \in \mathbb{N} \land b \in \mathbb{N}) \Rightarrow ($
$a \leq b \Leftrightarrow \mathrm{leq}(cast(a), cast(b)))$
equivalence over '≤ '

thm03: ⟨theorem⟩ $\forall x \cdot x \in \mathrm{ran}(cast) \Rightarrow (\exists i \cdot i \in \mathbb{N} \land cast^{-1}(x) = i)$

thm17: ⟨theorem⟩ $cast \in \mathbb{N} \rightarrowtail\!\!\!\!\twoheadrightarrow cast[\mathbb{N}]$

thm18: ⟨theorem⟩ $cast^{-1} \in cast[\mathbb{N}] \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbb{N}$

thm16: ⟨theorem⟩ $cast^{-1} \circ cast = \mathbb{N} \lhd id$

thm15: ⟨theorem⟩ $cast \circ cast^{-1} = cast[\mathbb{N}] \lhd id$

thm19: ⟨theorem⟩ $cast^{-1} \circ cast = \mathrm{dom}(cast) \lhd id$

thm20: ⟨theorem⟩ $cast \circ cast^{-1} = \mathrm{ran}(cast) \lhd id$

thm21: ⟨theorem⟩ $cast \in \mathrm{dom}(cast) \rightarrowtail\!\!\!\!\twoheadrightarrow \mathrm{ran}(cast)$
cast is a bijection

thm22: ⟨theorem⟩ $\forall n \cdot n \in \mathbb{N} \Rightarrow \text{leq}(zero, cast(n))$
$\forall$ n $\in$ $\mathbb{N}$, 0 $\leq$ cast(n)

END

CONTEXT C4_discrete
EXTENDS C2_margin
SETS

VT

CONSTANTS

tstep discrete time step

max_dp maximum delta for P during tstep

PBT

PV

AXIOMS

axm01: $tstep \in REAL\_STR\_POS$

axm03: $max\_dp \in REAL\_POS$
max variation of P during tstep

axm02: $\mathrm{leq}(max\_dp, z)$

axm04: $\mathrm{partition}(VT, \{PBT\}, \{PV\})$

tech01: ⟨theorem⟩ $\mathrm{leq}(zero, tstep)$

END

MACHINE M2_dsct_ctrl
REFINES M1_cntn_ctrl
SEES C3_cast,C4_discrete
VARIABLES

active

active_t

now

p abstract power value

pc continuous power function

pd discrete power function

i the current instant number

et time elapsed from previous discrete value sampling time

rs remaining continuous steps inside the discrete interval

nv next variant-related event type

INVARIANTS

type01: $pd \in 0 \,..\, i \rightarrow REAL\_POS$

type02: $i \in \mathbb{N}$

glue01: $\forall n \cdot n \in 0 \,..\, i \Rightarrow pc(cast(n) \text{ mult } tstep) = pd(n)$

n ∈ 0..i ⇒ pc(n*tstep) = pd(n)

glue02: $now = (cast(i) \text{ mult } tstep) \text{ plus } et$

now = i*tstep + et

prop02: $\forall n \cdot n \in 0 \,..\, i - 1 \Rightarrow ($
$\quad \forall t \cdot (\text{leq}(cast(n) \text{ mult } tstep, t)$
$\quad\quad \wedge \text{leq}(t, cast(n + 1) \text{ mult } tstep)) \Rightarrow ($
$\quad\quad \text{leq}(pd(n) \text{ sub } max\_dp, pc(t))$
$\quad\quad \wedge \text{leq}(pc(t), pd(n) \text{ plus } max\_dp)))$

∀ n < i, ∀ t ∈ [n*tstep , (n+1)*tstep], pd(n) − max_dp ≤ pc(t) ≤ pd(n) + max_dp

prop03: $\forall t \cdot (\text{leq}(cast(i) \text{ mult } tstep, t)$
$\quad \wedge \text{leq}(t, now)) \Rightarrow ($
$\quad \text{leq}(pd(i) \text{ sub } max\_dp, pc(t))$
$\quad \wedge \text{leq}(pc(t), pd(i) \text{ plus } max\_dp))$

∀ t ∈ [i*tstep , now], pd(n) − max_dp ≤ pc(t) ≤ pd(n) + max_dp

type03: $et \in REAL\_POS$

prop01: $\text{smr}(et, tstep)$

type04: $rs \in \mathbb{N}$

type05: $nv \in VT$

DLF_produce: ⟨theorem⟩ (
$\quad \exists dt \cdot ($
$\quad\quad (dt \in REAL\_STR\_POS)$
$\quad\quad \wedge (et = zero)$
$\quad\quad \wedge (\text{smr}(dt, tstep))$
$\quad)$
$) \vee ($
$\quad \exists dt \cdot ($
$\quad\quad (dt \in REAL\_STR\_POS)$
$\quad\quad \wedge (\text{smr}(zero, et))$
$\quad\quad \wedge (\text{smr}(et \text{ plus } dt, tstep))$
$\quad\quad \wedge (nv = PBT)$
$\quad\quad \wedge (rs > 0)$
$\quad)$
$) \vee ($

$(nv = PV)$
$\wedge (rs > 0)$
$) \vee ($
$\exists dt \cdot ($
$(dt \in REAL\_STR\_POS)$
$\wedge (et \text{ plus } dt = tstep)$
$\wedge (\text{smr}(zero, et))$
$\wedge (rs = 0)$
$)$
$)$
DLF on 'produce_*' events regarding dt,et,nv,rs

VARIANT

rs

EVENTS

Initialisation ⟨extended⟩

begin

act02: $active := \text{FALSE}$
act01: $p := zero$
act06: $now := zero$
act07: $pc := \lambda t \cdot t \in REAL\_POS | zero$
act08: $active\_t :\in REAL\_POS$
act09: $i := 0$
act11: $pd := \{0 \mapsto zero\}$
act12: $et := zero$
act13: $rs :\in \mathbb{N}$
no impact
act14: $nv :\in VT$
no impact

end

Event start ⟨ordinary⟩ $\widehat{=}$

extends start

any

dt
np
n_step
pd_start

where

grd02: $active = \text{FALSE}$
grd01: $p = zero$
grd04: $dt \in REAL\_STR\_POS$
dt > 0
grd05: $np \in REAL\_POS \nrightarrow REAL\_POS$
np ∈ R+ ⇸ R+
grd06: $\text{dom}(np) = \{t | t \in REAL \wedge \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt)\}$
dom(np) = [now,now+dt]
grd07: $np(now) = pc(now)$
np(now) = pc(now)
grd08: $np(now \text{ plus } dt) = m \text{ plus } z$
np(now+dt) = m+z
grd09: $\forall t1, t2 \cdot t1 \in \text{dom}(np) \wedge t2 \in \text{dom}(np) \wedge \text{smr}(t1, t2) \Rightarrow \text{smr}(np(t1), np(t2))$
np is a monotonically strictly increasing function : a<b ⇒ np(a)<np(b)
grd12: $\text{cnt\_int}(np, now, now \text{ plus } dt)$
np is continuous on [now,now+dt]

thm02: ⟨theorem⟩ $\mathrm{smr}(now, now \text{ plus } dt)$
thm01: ⟨theorem⟩ $\mathrm{dom}(pc \Leftarrow np) = REAL\_POS$
grd13: $n\_step \in \mathbb{N}_1$
grd19: $dt = cast(n\_step) \text{ mult } tstep$
grd14: $et = zero$
thm04: ⟨theorem⟩ $now = cast(i) \text{ mult } tstep$
thm03: ⟨theorem⟩ $now \text{ plus } dt = cast(i + n\_step) \text{ mult } tstep$
grd16: $pd\_start \in i \,..\, i + n\_step \rightarrow REAL\_POS$
thm05: ⟨theorem⟩ $\forall n \cdot n \in \mathbb{N} \Rightarrow$
$(n \in \mathrm{dom}(pd\_start) \Leftrightarrow cast(n) \text{ mult } tstep \in \mathrm{dom}(np))$
grd18: $pd\_start(i) = pd(i)$
grd17: $\forall n \cdot n \in \mathrm{dom}(pd\_start) \Rightarrow$
$np(cast(n) \text{ mult } tstep) = pd\_start(n)$
grd20: $\forall n \cdot n \in i \,..\, i + n\_step - 1 \Rightarrow ($
$\forall t \cdot (\mathrm{leq}(cast(n) \text{ mult } tstep, t)$
$\wedge \mathrm{leq}(t, cast(n+1) \text{ mult } tstep)) \Rightarrow$
$\mathrm{leq}(np(t), pd\_start(n) \text{ plus } max\_dp))$
thm06: ⟨theorem⟩ $\forall n \cdot n \in 0 \,..\, i - 1 \Rightarrow n \in \mathrm{dom}(pd) \wedge n \notin \mathrm{dom}(pd\_start)$
(pd⩤pd_start)(n), case 1/2: n < i
thm07: ⟨theorem⟩ $\forall n \cdot n \in i \,..\, i + n\_step - 1 \Rightarrow n \in \mathrm{dom}(pd\_start)$
(pd⩤pd_start)(n), case 2/2: n ≥ i
thm11: ⟨theorem⟩ $\forall n, t \cdot (n \in 0 \,..\, i - 1 \wedge t \neq now$
$\wedge \mathrm{leq}(cast(n) \text{ mult } tstep, t) \wedge \mathrm{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \mathrm{dom}(pc) \wedge t \notin \mathrm{dom}(np)$
(pc⩤np)(t), case 1/3: n < i ∧ t ≠ now
thm10: ⟨theorem⟩ $\forall n, t \cdot (n \in 0 \,..\, i - 1 \wedge t = now$
$\wedge \mathrm{leq}(cast(n) \text{ mult } tstep, t) \wedge \mathrm{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \mathrm{dom}(np)$
(pc⩤np)(t), case 2/3: n < i ∧ t = now
thm09: ⟨theorem⟩ $\forall n, t \cdot (n \in i \,..\, i + n\_step - 1$
$\wedge \mathrm{leq}(cast(n) \text{ mult } tstep, t) \wedge \mathrm{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \mathrm{dom}(np)$
(pc⩤np)(t), case 3/3: n ≥ i

then

act01: $active := \mathrm{TRUE}$
act02: $p := m \text{ plus } z$
act03: $now := now \text{ plus } dt$
act04: $pc := pc \Leftarrow np$
act05: $active\_t := now \text{ plus } dt$
act06: $i := i + n\_step$
act07: $pd := pd \Leftarrow pd\_start$

end

Event produce_from_tick ⟨ordinary⟩ $\widehat{=}$
extends produce_safe

any

new_p
dt
np

where

grd02: $active = \mathrm{TRUE}$
grd03: $new\_p \in REAL\_POS$
grd01: $\mathrm{leq}(m, new\_p) \wedge \mathrm{leq}(new\_p, M)$
grd10: $dt \in REAL\_STR\_POS$
dt > 0
thm02: ⟨theorem⟩ $\mathrm{smr}(now, now \text{ plus } dt)$

grd11: $np \in REAL\_POS \nrightarrow REAL\_POS$
np ∈ R+ ⇸ R+
grd06: $\mathrm{dom}(np) = \{t | t \in REAL \wedge \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt)\}$
dom(np) = [now,now+dt]
grd07: $np(now) = pc(now)$
np(now) = pc(now)
grd08: $np(now \text{ plus } dt) = new\_p$
np(now+dt) = new_p
grd09: $\mathrm{leq}(p, new\_p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t1), np(t2)))$
np is a monotonic function
grd14: $\mathrm{leq}(new\_p, p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t2), np(t1)))$
np is a monotonic function
grd12: $\mathrm{cnt\_int}(np, now, now \text{ plus } dt)$
np is continuous on [now,now+dt]
thm01: ⟨theorem⟩ $\mathrm{dom}(pc \Leftarrow np) = REAL\_POS$
grd13: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(m \text{ plus } z, np(t)) \wedge \mathrm{leq}(np(t), M \text{ sub } z)$
∀ t ∈ [now,now+dt] ⇒ np(t) ∈ [m+z,M−z]
grd15: $et = zero$
grd17: $\mathrm{smr}(dt, tstep)$
grd16: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(pd(i) \text{ sub } max\_dp, np(t)) \wedge \mathrm{leq}(np(t), pd(i) \text{ plus } max\_dp)$

physical assumption

then

act01: $p := new\_p$
act02: $now := now \text{ plus } dt$
act03: $pc := pc \Leftarrow np$
act06: $et := et \text{ plus } dt$
act07: $rs :\in \mathbb{N}$
act08: $nv := PBT$

end

Event produce_between_ticks ⟨ordinary⟩ $\widehat{=}$
extends produce_safe

any

new_p
dt
np

where

grd02: $active = \mathrm{TRUE}$
grd03: $new\_p \in REAL\_POS$
grd01: $\mathrm{leq}(m, new\_p) \wedge \mathrm{leq}(new\_p, M)$
grd10: $dt \in REAL\_STR\_POS$
dt > 0
thm02: ⟨theorem⟩ $\mathrm{smr}(now, now \text{ plus } dt)$
grd11: $np \in REAL\_POS \nrightarrow REAL\_POS$
np ∈ R+ ⇸ R+
grd06: $\mathrm{dom}(np) = \{t | t \in REAL \wedge \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt)\}$
dom(np) = [now,now+dt]
grd07: $np(now) = pc(now)$
np(now) = pc(now)
grd08: $np(now \text{ plus } dt) = new\_p$
np(now+dt) = new_p
grd09: $\mathrm{leq}(p, new\_p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t1), np(t2)))$

np is a monotonic function

grd14: $\mathrm{leq}(new\_p, p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \land t2 \in \mathrm{dom}(np) \land \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t2), np(t1)))$

np is a monotonic function

grd12: $\mathrm{cnt\_int}(np, now, now \text{ plus } dt)$

np is continuous on [now,now+dt]

thm01: ⟨theorem⟩ $\mathrm{dom}(pc \Leftarrow np) = REAL\_POS$

grd13: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(m \text{ plus } z, np(t)) \land \mathrm{leq}(np(t), M \text{ sub } z)$

∀ t ∈ [now,now+dt] ⇒ np(t) ∈ [m+z,M−z]

grd18: $\mathrm{smr}(zero, et)$

grd15: $\mathrm{smr}(et \text{ plus } dt, tstep)$

grd16: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(pd(i) \text{ sub } max\_dp, np(t)) \land \mathrm{leq}(np(t), pd(i) \text{ plus } max\_dp)$

physical assumption

grd17: $nv = PBT$

grd19: $rs > 0$

then

act01: $p := new\_p$

act02: $now := now \text{ plus } dt$

act03: $pc := pc \Leftarrow np$

act04: $et := et \text{ plus } dt$

act05: $nv := PV$

end

Event produce_variant ⟨convergent⟩ $\widehat{=}$

when

grd01: $nv = PV$

grd02: $rs > 0$

then

act01: $rs :| rs' \in \mathbb{N} \land rs' < rs$

act02: $nv := PBT$

end

Event produce_on_tick ⟨ordinary⟩ $\widehat{=}$

extends produce_safe

any

new_p

dt

np

where

grd02: $active = \mathrm{TRUE}$

grd03: $new\_p \in REAL\_POS$

grd01: $\mathrm{leq}(m, new\_p) \land \mathrm{leq}(new\_p, M)$

grd10: $dt \in REAL\_STR\_POS$

dt > 0

thm02: ⟨theorem⟩ $\mathrm{smr}(now, now \text{ plus } dt)$

grd11: $np \in REAL\_POS \nrightarrow REAL\_POS$

np ∈ R+ ⇸ R+

grd06: $\mathrm{dom}(np) = \{t | t \in REAL \land \mathrm{leq}(now, t) \land \mathrm{leq}(t, now \text{ plus } dt)\}$

dom(np) = [now,now+dt]

grd07: $np(now) = pc(now)$

np(now) = pc(now)

grd08: $np(now \text{ plus } dt) = new\_p$

np(now+dt) = new_p

grd09: $\mathrm{leq}(p, new\_p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \land t2 \in \mathrm{dom}(np) \land \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t1), np(t2)))$

np is a monotonic function

grd14: $\mathrm{leq}(new\_p, p) \Rightarrow (\forall t1, t2 \cdot t1 \in \mathrm{dom}(np) \wedge t2 \in \mathrm{dom}(np) \wedge \mathrm{leq}(t1, t2) \Rightarrow \mathrm{leq}(np(t2), np(t1)))$
np is a monotonic function
grd12: $\mathrm{cnt\_int}(np, now, now \text{ plus } dt)$
np is continuous on [now,now+dt]
thm01: ⟨theorem⟩ $\mathrm{dom}(pc \Leftarrow np) = REAL\_POS$
grd13: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(m \text{ plus } z, np(t)) \wedge \mathrm{leq}(np(t), M \text{ sub } z)$
∀ t ∈ [now,now+dt] ⇒ np(t) ∈ [m+z,M−z]
grd15: $et \text{ plus } dt = tstep$
grd18: $\mathrm{smr}(zero, et)$
grd17: $rs = 0$
thm03: ⟨theorem⟩ $cast(i + 1) \text{ mult } tstep = now \text{ plus } dt$
grd16: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(pd(i) \text{ sub } max\_dp, np(t)) \wedge \mathrm{leq}(np(t), pd(i) \text{ plus } max\_dp)$

physical assumption

then

act01: $p := new\_p$
act02: $now := now \text{ plus } dt$
act03: $pc := pc \Leftarrow np$
act04: $i := i + 1$
act05: $pd(i + 1) := new\_p$
act06: $et := zero$

end

Event safety_stop ⟨ordinary⟩ $\widehat{=}$

pd(i) is in the safe zone (now)
pd(i+1) is not in the safe zone (safety risk)
pd(i+n_step)=0

extends safety_stop

any

dt
np
n_step
pd_stop

where

grd02: $active = \mathrm{TRUE}$
grd01: $\mathrm{leq}(m, p) \wedge \mathrm{leq}(p, M)$
grd10: $dt \in REAL\_STR\_POS$
dt > 0
thm02: ⟨theorem⟩ $\mathrm{smr}(now, now \text{ plus } dt)$
grd11: $np \in REAL\_POS \nrightarrow REAL\_POS$
np ∈ R+ ⇸ R+
grd06: $\mathrm{dom}(np) = \{t | t \in REAL \wedge \mathrm{leq}(now, t) \wedge \mathrm{leq}(t, now \text{ plus } dt)\}$
dom(np) = [now,now+dt]
grd07: $np(now) = pc(now)$
np(now) = pc(now)
grd08: $np(now \text{ plus } dt) = zero$
np(now+dt) = 0
grd12: $\mathrm{cnt\_int}(np, now, now \text{ plus } dt)$
np is continuous on [now,now+dt]
thm01: ⟨theorem⟩ $\mathrm{dom}(pc \Leftarrow np) = REAL\_POS$
grd53: $\forall t \cdot t \in \mathrm{dom}(np) \Rightarrow \mathrm{leq}(np(t), M)$
∀ t ∈ [now,now+dt] ⇒ np(t) ≤ M
grd54: $\exists t \cdot t \in \mathrm{dom}(np) \Rightarrow \neg(\mathrm{leq}(m \text{ plus } z, np(t)) \wedge \mathrm{leq}(np(t), M \text{ sub } z))$
∃ t ∈ [now,now+dt] ⇒ ¬ np(t) ∈ [m+z,M−z] ; safety risk
grd33: $n\_step \geq 2$

grd13: ⟨theorem⟩ $n\_step \in \mathbb{N}_1$
grd19: $dt = cast(n\_step) \text{ mult } tstep$
grd14: $et = zero$
thm03: ⟨theorem⟩ $now = cast(i) \text{ mult } tstep$
thm04: ⟨theorem⟩ $now \text{ plus } dt = cast(i + n\_step) \text{ mult } tstep$
grd16: $pd\_stop \in i \,..\, i + n\_step \rightarrow REAL\_POS$
thm05: ⟨theorem⟩ $\forall n \cdot n \in \mathbb{N} \Rightarrow$
$(n \in \text{dom}(pd\_stop) \Leftrightarrow cast(n) \text{ mult } tstep \in \text{dom}(np))$
grd18: $pd\_stop(i) = pd(i)$
grd17: $\forall n \cdot n \in \text{dom}(pd\_stop) \Rightarrow$
$np(cast(n) \text{ mult } tstep) = pd\_stop(n)$
grd20: $\forall n \cdot n \in i \,..\, i + n\_step - 1 \Rightarrow ($
$\forall t \cdot (\text{leq}(cast(n) \text{ mult } tstep, t)$
$\wedge \text{leq}(t, cast(n+1) \text{ mult } tstep)) \Rightarrow ($
$\text{leq}(pd\_stop(n) \text{ sub } max\_dp, np(t))$
$\wedge \text{leq}(np(t), pd\_stop(n) \text{ plus } max\_dp)))$
grd21: $\text{smr}(pd\_stop(i+1), m \text{ plus } z) \vee \text{gtr}(pd\_stop(i+1), M \text{ sub } z)$
(pd_stop(i+1) < m + z) ∨ (pd_stop(i+1) > M−z)
grd09: $\forall t1, t2 \cdot \text{leq}(now \text{ plus } tstep, t1) \wedge t1 \in \text{dom}(np) \wedge t2 \in \text{dom}(np) \wedge \text{smr}(t1, t2) \Rightarrow$
$\text{gtr}(np(t1), np(t2))$
np is a monotonically strictly decreasing function after the next discrete instant :
(now+tstep≤ a ∧ a<b) ⇒ np(a)>np(b)
thm06: ⟨theorem⟩ $\forall n \cdot n \in 0 \,..\, i - 1 \Rightarrow n \in \text{dom}(pd) \wedge n \notin \text{dom}(pd\_stop)$
(pd⩤pd_stop)(n), case 1/2: n < i
thm07: ⟨theorem⟩ $\forall n \cdot n \in i \,..\, i + n\_step - 1 \Rightarrow n \in \text{dom}(pd\_stop)$
(pd⩤pd_stop)(n), case 2/2: n ≥ i
thm11: ⟨theorem⟩ $\forall n, t \cdot (n \in 0 \,..\, i - 1 \wedge t \neq now$
$\wedge \text{leq}(cast(n) \text{ mult } tstep, t) \wedge \text{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \text{dom}(pc) \wedge t \notin \text{dom}(np)$
(pc⩤np)(t), case 1/3: n < i ∧ t ≠ now
thm10: ⟨theorem⟩ $\forall n, t \cdot (n \in 0 \,..\, i - 1 \wedge t = now$
$\wedge \text{leq}(cast(n) \text{ mult } tstep, t) \wedge \text{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \text{dom}(np)$
(pc⩤np)(t), case 2/3: n < i ∧ t = now
thm09: ⟨theorem⟩ $\forall n, t \cdot (n \in i \,..\, i + n\_step - 1$
$\wedge \text{leq}(cast(n) \text{ mult } tstep, t) \wedge \text{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \text{dom}(np)$
(pc⩤np)(t), case 3/3: n ≥ i

then

act01: $active := \text{FALSE}$
act02: $p := zero$
act03: $now := now \text{ plus } dt$
act04: $pc := pc \Leftarrow np$
act05: $i := i + n\_step$
act06: $pd := pd \Leftarrow pd\_stop$

end

Event stop ⟨ordinary⟩ $\widehat{=}$
extends stop

any

np
dt
n_step
pd_stop

where

grd02: $active = \text{TRUE}$

grd01: $\text{leq}(m, p) \wedge \text{leq}(p, M)$

grd04: $dt \in REAL\_STR\_POS$

dt > 0

thm02: ⟨theorem⟩ $\text{smr}(now, now \text{ plus } dt)$

grd05: $np \in REAL\_POS \nrightarrow REAL\_POS$

np ∈ R+ ⇸ R+

grd06: $\text{dom}(np) = \{t | t \in REAL \wedge \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt)\}$

dom(np) = [now,now+dt]

grd07: $np(now) = pc(now)$

np(now) = pc(now)

grd08: $np(now \text{ plus } dt) = zero$

np(now+dt) = 0

grd09: $\forall t1, t2 \cdot t1 \in \text{dom}(np) \wedge t2 \in \text{dom}(np) \wedge \text{smr}(t1, t2) \Rightarrow$
$\text{gtr}(np(t1), np(t2))$

np is a monotonically strictly decreasing function : a<b ⇒ np(a)>np(b)

grd12: $\text{cnt\_int}(np, now, now \text{ plus } dt)$

np is continuous on [now,now+dt]

thm01: ⟨theorem⟩ $\text{dom}(pc \Leftarrow np) = REAL\_POS$

grd13: $n\_step \in \mathbb{N}_1$

grd19: $dt = cast(n\_step) \text{ mult } tstep$

grd14: $et = zero$

thm03: ⟨theorem⟩ $now = cast(i) \text{ mult } tstep$

thm04: ⟨theorem⟩ $now \text{ plus } dt = cast(i + n\_step) \text{ mult } tstep$

grd16: $pd\_stop \in i \,..\, i + n\_step \rightarrow REAL\_POS$

thm05: ⟨theorem⟩ $\forall n \cdot n \in \mathbb{N} \Rightarrow$
$(n \in \text{dom}(pd\_stop) \Leftrightarrow cast(n) \text{ mult } tstep \in \text{dom}(np))$

grd18: $pd\_stop(i) = pd(i)$

grd17: $\forall n \cdot n \in \text{dom}(pd\_stop) \Rightarrow$
$np(cast(n) \text{ mult } tstep) = pd\_stop(n)$

grd20: $\forall n \cdot n \in i \,..\, i + n\_step - 1 \Rightarrow ($
$\forall t \cdot (\text{leq}(cast(n) \text{ mult } tstep, t)$
$\wedge \text{leq}(t, cast(n+1) \text{ mult } tstep)) \Rightarrow$
$\text{leq}(pd\_stop(n) \text{ sub } max\_dp, np(t)))$

thm06: ⟨theorem⟩ $\forall n \cdot n \in 0 \,..\, i - 1 \Rightarrow n \in \text{dom}(pd) \wedge n \notin \text{dom}(pd\_stop)$

(pd⩤pd_stop)(n), case 1/2: n < i

thm07: ⟨theorem⟩ $\forall n \cdot n \in i \,..\, i + n\_step - 1 \Rightarrow n \in \text{dom}(pd\_stop)$

(pd⩤pd_stop)(n), case 2/2: n ≥ i

thm11: ⟨theorem⟩ $\forall n, t \cdot (n \in 0 \,..\, i - 1 \wedge t \neq now$
$\wedge \text{leq}(cast(n) \text{ mult } tstep, t) \wedge \text{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \text{dom}(pc) \wedge t \notin \text{dom}(np)$

(pc⩤np)(t), case 1/3: n < i ∧ t ≠ now

thm10: ⟨theorem⟩ $\forall n, t \cdot (n \in 0 \,..\, i - 1 \wedge t = now$
$\wedge \text{leq}(cast(n) \text{ mult } tstep, t) \wedge \text{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \text{dom}(np)$

(pc⩤np)(t), case 2/3: n < i ∧ t = now

thm09: ⟨theorem⟩ $\forall n, t \cdot (n \in i \,..\, i + n\_step - 1$
$\wedge \text{leq}(cast(n) \text{ mult } tstep, t) \wedge \text{leq}(t, cast(n+1) \text{ mult } tstep))$
$\Rightarrow t \in \text{dom}(np)$

(pc⩤np)(t), case 3/3: n ≥ i

then

act01: $active := \text{FALSE}$

act02: $p := zero$

act03: $now := now \text{ plus } dt$

act04: $pc := pc \Leftarrow np$

act05: $i := i + n\_step$

act06: $pd := pd \ntriangleleft pd\_stop$

end

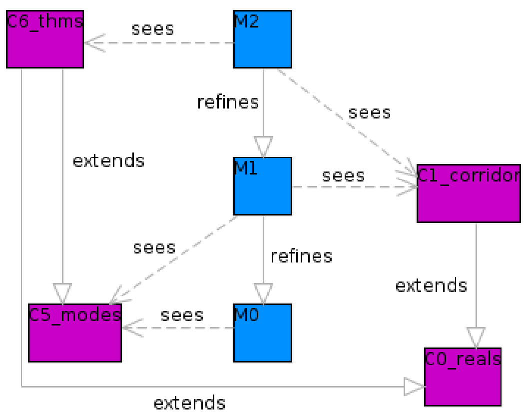
END

D Hybrid systems: Substitution

Components:

- *C5* modes (page 246)
- *C0* properties on reals (page 247)
- *C1* envelope (page 250)
- *C6* some technical theorems (page 251)
- *M0* modes (page 252)
- *M1* f, g, p (page 254)
- *M2* $f(t)$, $g(t)$, $p(t)$ (page 257)

Theory used in this development: *Real* (page 140)



The models are also available at: `http://babin.perso.enseeiht.fr/r/thesis/`

CONTEXT C5_modes
SETS
MODES
CONSTANTS
MODE_F
MODE_R
MODE_G
AXIOMS
axm1: partition($MODES, \{MODE\_F\}, \{MODE\_R\}, \{MODE\_G\}$)
END

CONTEXT C0_reals

theorems concerning continuous mathematical functions

CONSTANTS

REAL_POS

REAL_STR_POS

AXIOMS

def01: $REAL\_POS = \{x|x \in REAL \land \mathrm{leq}(zero, x)\}$

def02: $REAL\_STR\_POS = \{x|x \in REAL \land \mathrm{smr}(zero, x)\}$

thm01: ⟨theorem⟩ $REAL\_POS \subseteq REAL$

thm02: ⟨theorem⟩ $REAL\_STR\_POS \subseteq REAL\_POS$

thm03: ⟨theorem⟩ $REAL\_STR\_POS \subseteq REAL$

thm39: ⟨theorem⟩ $\forall a, b \cdot a \in REAL \land b \in REAL \Rightarrow (a = a \,\mathrm{plus}\, b \Rightarrow b = zero)$

thm04: ⟨theorem⟩ $zero \in REAL\_POS$

thm05: ⟨theorem⟩ $\mathrm{leq}(zero, zero)$

thm06: ⟨theorem⟩ $\forall n, A, f, a \cdot n \in \mathbb{N}$
$\land A \subseteq REAL$
$\land f \in 0 \,..\, n \to A$
$\land a \in A$
$\Rightarrow f \cup \{n + 1 \mapsto a\} \in 0 \,..\, n + 1 \to A$

thm07: ⟨theorem⟩ $\forall a, b, c \cdot (a \in REAL \land b \in REAL \land c \in REAL) \Rightarrow$
$(\mathrm{leq}(a \,\mathrm{plus}\, c, b \,\mathrm{plus}\, c) \Leftrightarrow \mathrm{leq}(a, b))$

a+c ≤ b+c ⇔ a≤ b

thm08: ⟨theorem⟩ $\forall x \cdot x \in REAL \Rightarrow$
$(\mathrm{leq}(zero, x) \Leftrightarrow \mathrm{leq}(\mathrm{minus}(x), zero))$

0≤ x ⇔ -x≤ 0

thm09: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(a, b) \Leftrightarrow \mathrm{leq}(zero, b \,\mathrm{sub}\, a))$

a≤ b ⇔ 0≤ b-a

thm10: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(zero, a) \Leftrightarrow \mathrm{leq}(b, b \,\mathrm{plus}\, a))$

0≤ a ⇔ b≤ b+a

thm11: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(zero, b) \Rightarrow \mathrm{leq}(a, a \,\mathrm{plus}\, b))$

0≤ b ⇒ a ≤ a+b

thm14: ⟨theorem⟩ $\forall a, b \cdot a \in REAL \land b \in REAL \Rightarrow$
$(a = b \Leftrightarrow b = a)$

a=b ⇔ b=a

thm13: ⟨theorem⟩ $\forall a, b \cdot a \in REAL \land b \in REAL \Rightarrow$
$(\neg(a = b) \Leftrightarrow \neg(b = a))$

¬(a=b) ⇔ ¬(b=a)

thm12: ⟨theorem⟩ $\forall a, b \cdot (a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{smr}(zero, b) \Rightarrow \mathrm{smr}(a, a \,\mathrm{plus}\, b))$

0<b ⇒ a < a+b

thm33: ⟨theorem⟩ $\forall a \cdot zero \,\mathrm{mult}\, a = zero$

0 * a = 0

thm38: ⟨theorem⟩ $\forall a \cdot a \,\mathrm{mult}\, \mathrm{minus}(one) = \mathrm{minus}(a)$

a*(−1) = −a

thm41: ⟨theorem⟩ $\forall a \cdot \mathrm{minus}(\mathrm{minus}(a)) = a$

−(−a) = a

thm17: ⟨theorem⟩ $\mathrm{leq}(zero, one)$

0≤ 1

thm15: ⟨theorem⟩ $\mathrm{smr}(zero, one)$
0<1

thm34: ⟨theorem⟩ $\forall a, b\cdot(a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(zero, b) \Rightarrow \mathrm{leq}(a \,\mathrm{sub}\, b, a))$
$0 \leq b \Rightarrow a\text{-}b \leq a$

thm16: ⟨theorem⟩ $\forall a, b\cdot(a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{smr}(zero, b) \Rightarrow \mathrm{smr}(a \,\mathrm{sub}\, b, a))$
$0<b \Rightarrow a\text{-}b < a$

thm20: ⟨theorem⟩ $\forall a, b\cdot(a \in REAL \land b \in REAL) \Rightarrow$
$(\mathrm{leq}(a, b) \land \mathrm{leq}(b, a) \Leftrightarrow a = b)$
$a \leq b \land b \leq a \Leftrightarrow a=b$

thm21: ⟨theorem⟩ $\forall a, b\cdot(a \in REAL \land b \in REAL) \Rightarrow$
$(\neg\mathrm{leq}(a, b) \Leftrightarrow \mathrm{gtr}(a, b))$
$\neg(a \leq b) \Leftrightarrow a>b$

thm22: $\forall a, b\cdot(a \in REAL\_POS \land b \in REAL\_POS) \Rightarrow$
$(a \,\mathrm{mult}\, b \in REAL\_POS)$
$a \in R+ \land b \in R+ \Rightarrow a*b \in R+$

thm23: $\forall a, b\cdot(a \in REAL \land b \in REAL) \Rightarrow$
$((\exists c\cdot c \in REAL\_STR\_POS \land a = b \,\mathrm{plus}\, c) \Leftrightarrow \mathrm{smr}(b, a))$
$(\exists\ c > 0,\ a = b+c) \Leftrightarrow b<a$

thm24: $\forall a, b, c\cdot(a \in REAL \land b \in REAL \land c \in REAL) \Rightarrow$
$(\mathrm{smr}(a, b) \land \mathrm{smr}(b, c) \Rightarrow \mathrm{smr}(a, c))$
$a<b \land b<c \Rightarrow a<c$

thm26: ⟨theorem⟩ $\forall a, b, c\cdot(a \in REAL \land b \in REAL \land c \in REAL) \Rightarrow$
$(\mathrm{leq}(a, b) \land \mathrm{smr}(b, c) \Rightarrow \mathrm{smr}(a, c))$
$a \leq b \land b<c \Rightarrow a<c$

thm27: ⟨theorem⟩ $\forall a, b\cdot\mathrm{leq}(a, b) \lor \mathrm{leq}(b, a)$
$a \leq b \lor b \leq a$

thm30: $\forall a, b, c\cdot(a \in REAL\_POS \land b \in REAL\_POS \land c \in REAL\_STR\_POS) \Rightarrow$
$(\mathrm{smr}(a, b) \Rightarrow \mathrm{smr}(a \,\mathrm{mult}\, c, b \,\mathrm{mult}\, c))$
$a \geq 0 \land b \geq 0 \land c > 0 \Rightarrow (a < b \Rightarrow a*c < b*c)$

thm31: $\forall a, b, c\cdot(a \in REAL\_POS \land b \in REAL\_POS \land c \in REAL\_POS) \Rightarrow$
$(\mathrm{leq}(a, b) \Rightarrow \mathrm{leq}(a \,\mathrm{mult}\, c, b \,\mathrm{mult}\, c))$
$a \geq 0 \land b \geq 0 \land c \geq 0 \Rightarrow (a \leq b \Rightarrow a*c \leq b*c)$

thm40: $\forall a, b, c\cdot(a \in REAL\_POS \land b \in REAL\_POS \land c \in REAL\_STR\_POS) \Rightarrow$
$(\mathrm{leq}(a \,\mathrm{mult}\, c, b \,\mathrm{mult}\, c) \Rightarrow \mathrm{leq}(a, b))$
$a \geq 0 \land b \geq 0 \land c > 0 \Rightarrow (a*c \leq b*c \Rightarrow a \leq b)$

thm32: ⟨theorem⟩ $\forall a, b\cdot\mathrm{smr}(a, b) \Leftrightarrow \neg\mathrm{leq}(b, a)$
$a<b \Leftrightarrow \neg\ b \leq a$

thm35: $\forall a\cdot a \in REAL\_STR\_POS \Rightarrow ($
$\exists b\cdot b \in REAL\_STR\_POS \land \mathrm{smr}(b, a))$
$\forall\ a > 0, \exists\ b\ /\ 0 < b < a$

thm36: $\forall a, b\cdot\mathrm{smr}(a, b) \Leftrightarrow \mathrm{smr}(zero, b \,\mathrm{sub}\, a)$
$a < b \Leftrightarrow 0 < b - a$

thm37: $\forall a, b, c\cdot\mathrm{smr}(a, b) \Leftrightarrow \mathrm{smr}(a \,\mathrm{plus}\, c, b \,\mathrm{plus}\, c)$
$a < b \Leftrightarrow a+c < b+c$

thm42: ⟨theorem⟩ $\forall a, b, f, g\cdot$
$a \in REAL\_POS$
$\land\ \mathrm{leq}(a, b)$
$\land\ f \in \{t|\mathrm{leq}(zero, t) \land \mathrm{leq}(t, a)\} \to REAL\_POS$
$\land\ g \in \{t|\mathrm{leq}(a, t) \land \mathrm{leq}(t, b)\} \to REAL\_POS$
$\Rightarrow$
$f \Lleftarrow g \in \{t|\mathrm{leq}(zero, t) \land \mathrm{leq}(t, b)\} \to REAL\_POS$

thm43: $\forall a, b, c, f \cdot$

$$a \in REAL\_POS$$
$$\land \text{leq}(a, b)$$
$$\land f \in \{t | \text{leq}(zero, t) \land \text{leq}(t, a)\} \rightarrow REAL\_POS$$

$\Rightarrow$

$$(f \lhd\!\!\!+ (\lambda t \cdot \text{leq}(a, t) \land \text{leq}(t, b) | c))(b) = c$$

END

CONTEXT C1_corridor

energy corridor

EXTENDS C0_reals

CONSTANTS

m

M

AXIOMS

axm01: $m \in REAL\_STR\_POS$

axm02: $M \in REAL\_STR\_POS$

axm03: $\mathrm{smr}(m, M)$

thm01: ⟨theorem⟩ $\mathrm{leq}(m, M)$

thm02: ⟨theorem⟩ $\mathrm{leq}(zero, m)$

thm06: ⟨theorem⟩ $\mathrm{leq}(zero, M)$

thm03: ⟨theorem⟩ $\forall x \cdot \mathrm{leq}(m, x) \Rightarrow x \in REAL\_POS$

thm04: ⟨theorem⟩ $\mathrm{leq}(m, m)$

thm05: ⟨theorem⟩ $\forall a \cdot \mathrm{leq}(m, a) \Rightarrow \mathrm{leq}(zero, a)$

END

CONTEXT C6_thms
EXTENDS C0_reals,C5_modes
AXIOMS

`thm01`: ⟨theorem⟩ $\forall a, b, f, g \cdot$

$$
\begin{aligned}
& \quad a \in REAL\_POS \\
& \land \operatorname{leq}(a, b) \\
& \land f \in \{t \mid \operatorname{leq}(zero, t) \land \operatorname{leq}(t, a)\} \to MODES \\
& \land g \in \{t \mid \operatorname{leq}(a, t) \land \operatorname{leq}(t, b)\} \to MODES \\
\Rightarrow & \\
& f \mathbin{\lhd\!\!\!-} g \in \{t \mid \operatorname{leq}(zero, t) \land \operatorname{leq}(t, b)\} \to MODES
\end{aligned}
$$

`thm02`: $\forall a, b, c, f \cdot$

$$
\begin{aligned}
& \quad a \in REAL\_POS \\
& \land \operatorname{leq}(a, b) \\
& \land f \in \{t \mid \operatorname{leq}(zero, t) \land \operatorname{leq}(t, a)\} \to MODES \\
\Rightarrow & \\
& (f \mathbin{\lhd\!\!\!-} (\lambda t \cdot \operatorname{leq}(a, t) \land \operatorname{leq}(t, b) \mid c))(b) = c
\end{aligned}
$$

END

MACHINE M0
SEES C5_modes
VARIABLES

active active is true once the system has started

md the mode of the system

INVARIANTS

type01: $active \in \mathrm{BOOL}$

type03: $md \in MODES$

tech01: $active = \mathrm{FALSE} \Rightarrow md = MODE\_F$

DLF: ⟨theorem⟩ (
$(active = \mathrm{FALSE})$
$\wedge (md = MODE\_F)$
) $\vee$ (
$(active = \mathrm{FALSE})$
$\wedge (md = MODE\_F)$
) $\vee$ (
$(active = \mathrm{TRUE})$
$\wedge (md = MODE\_F \vee md = MODE\_G)$
) $\vee$ (
$(active = \mathrm{TRUE})$
$\wedge (md = MODE\_F)$
) $\vee$ (
$(active = \mathrm{TRUE})$
$\wedge (md = MODE\_R)$
) $\vee$ (
$(active = \mathrm{TRUE})$
$\wedge (md = MODE\_R)$
)

EVENTS
Initialisation
begin
act1: $active := \mathrm{FALSE}$
act3: $md := MODE\_F$
end
Event boot ⟨ordinary⟩ $\widehat{=}$
when
grd1: $active = \mathrm{FALSE}$
grd2: $md = MODE\_F$
then
skip
end
Event start ⟨ordinary⟩ $\widehat{=}$
when
grd1: $active = \mathrm{FALSE}$
grd2: $md = MODE\_F$
then
act1: $active := \mathrm{TRUE}$
end
Event progress ⟨ordinary⟩ $\widehat{=}$
when
grd2: $active = \mathrm{TRUE}$
grd1: $md = MODE\_F \vee md = MODE\_G$

then

skip

end

Event fail_f ⟨ordinary⟩ $\widehat{=}$

when

grd2: $active = \mathrm{TRUE}$

grd1: $md = MODE\_F$

then

act1: $md := MODE\_R$

end

Event repair ⟨ordinary⟩ $\widehat{=}$

when

grd2: $active = \mathrm{TRUE}$

grd1: $md = MODE\_R$

then

skip

end

Event repaired_g ⟨ordinary⟩ $\widehat{=}$

when

grd2: $active = \mathrm{TRUE}$

grd1: $md = MODE\_R$

then

act1: $md := MODE\_G$

end

END

MACHINE M1
REFINES M0
SEES C1_corridor,C5_modes
VARIABLES

active
  [refined]
  (should only be modified by CTRL events)

md
  [refined]
  (should only be modified by CTRL events)

p
  p is the amount of power produced by the system
  (should only be modified by ENV events)

f (should only be modified by ENV events)

g (should only be modified by ENV events)

INVARIANTS

type02: $p \in REAL\_POS$
type04: $f \in REAL\_POS$
type05: $g \in REAL\_POS$
corridor01: $\mathrm{leq}(p, M)$
  p ≤ M
corridor02: $active = \mathrm{TRUE} \Rightarrow \mathrm{leq}(m, p)$
  active ⇒ m ≤ p
mode01: $md = MODE\_F \Rightarrow p = f$
mode04: $md = MODE\_F \Rightarrow g = zero$
mode02: $md = MODE\_R \Rightarrow p = f \text{ plus } g$
mode03: $md = MODE\_G \Rightarrow p = g$
mode05: $md = MODE\_G \Rightarrow f = zero$
thm01: ⟨theorem⟩ $p = f \text{ plus } g$
thm02: ⟨theorem⟩ $\mathrm{leq}(f, M)$
  f ≤ M
thm03: ⟨theorem⟩ $\mathrm{leq}(g, M)$
  g ≤ M

EVENTS

Initialisation ⟨extended⟩

begin
  act1: $active := \mathrm{FALSE}$
  act3: $md := MODE\_F$
  act2: $p := zero$
  act4: $f := zero$
  act5: $g := zero$
end

Event ENV_starting_f ⟨ordinary⟩ $\widehat{=}$
extends boot

any
  new_f
where
  grd1: $active = \mathrm{FALSE}$
  grd2: $md = MODE\_F$
  grd4: $\mathrm{leq}(f, new\_f)$
    f ≤ new_f (f is increasing)

grd3: $\text{leq}(new\_f, M)$
new_f ≤ M
then
act1: $f := new\_f$
act2: $p := new\_f$
end
Event CTRL_started ⟨ordinary⟩ $\widehat{=}$
extends start
when
grd1: $active = \text{FALSE}$
grd2: $md = MODE\_F$
grd3: $\text{leq}(m, p)$
grd4: $\text{leq}(p, M)$
then
act1: $active := \text{TRUE}$
end
Event ENV_evolution_f ⟨ordinary⟩ $\widehat{=}$
refines progress
any
new_f
where
grd2: $active = \text{TRUE}$
grd1: $md = MODE\_F$
grd5: $f \neq m$
grd6: $f \neq M$
grd3: $\text{leq}(m, new\_f)$
m ≤ new_f
grd4: $\text{leq}(new\_f, M)$
new_f ≤ M
then
act1: $f := new\_f$
act2: $p := new\_f$
end
Event CTRL_limit_detected_f ⟨ordinary⟩ $\widehat{=}$
extends fail_f
when
grd2: $active = \text{TRUE}$
grd1: $md = MODE\_F$
grd5: $f = m \lor f = M$
then
act1: $md := MODE\_R$
end
Event ENV_evolution_fg ⟨ordinary⟩ $\widehat{=}$
extends repair
any
new_f
new_g
where
grd2: $active = \text{TRUE}$
grd1: $md = MODE\_R$
grd3: $\text{leq}(m, new\_f \text{ plus } new\_g)$
m ≤ new_f + new_g
grd4: $\text{leq}(new\_f \text{ plus } new\_g, M)$
new_f + new_g ≤ M

grd5: leq($zero, new\_f$)
0 ≤ new_f
grd6: leq(new_f, f)
new_f ≤ f (f is decreasing)
grd7: leq(g, new_g)
g ≤ new_g (g is increasing)
grd8: leq(new_g, M)
new_g ≤ M

then

act1: $f := new\_f$
act2: $g := new\_g$
act3: $p := new\_f$ plus new_g

end

Event CTRL_repaired_g ⟨ordinary⟩ $\widehat{=}$
extends repaired_g

when

grd2: $active =$ TRUE
grd1: $md = MODE\_R$
grd3: leq(m, g)
m ≤ g
grd4: leq(g, M)
g ≤ M
grd5: $f = zero$
so that going from 'f+g' to 'g' is continuous

then

act1: $md := MODE\_G$

end

Event ENV_evolution_g ⟨ordinary⟩ $\widehat{=}$
refines progress

any

new_g

where

grd2: $active =$ TRUE
grd1: $md = MODE\_G$
grd3: leq(m, new_g)
m ≤ new_g
grd4: leq(new_g, M)
new_g ≤ M

then

act1: $g := new\_g$
act2: $p := new\_g$

end

END

MACHINE M2
REFINES M1
SEES C1_corridor,C6_thms
VARIABLES

active [refined]

active_t

has a sense only if active is TRUE
time (moment) when active became true
(should only be modified by CTRL events)

md [refined]

md_c

now (should only be modified by ENV events)

p_c (should only be modified by ENV events)

f_c (should only be modified by ENV events)

g_c (should only be modified by ENV events)

INVARIANTS

type01: $now \in REAL\_POS$

type06: $active\_t \in REAL\_POS$

type02: $p\_c \in \{t | \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now)\} \rightarrow REAL\_POS$

type03: $f\_c \in \{t | \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now)\} \rightarrow REAL\_POS$

type04: $g\_c \in \{t | \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now)\} \rightarrow REAL\_POS$

type05: $md\_c \in \{t | \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now)\} \rightarrow MODES$

mode02: $\forall t \cdot \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now) \land md\_c(t) = MODE\_R \Rightarrow p\_c(t) = f\_c(t) \text{ plus } g\_c(t)$

mode01: $\forall t \cdot \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now) \land md\_c(t) = MODE\_F \Rightarrow p\_c(t) = f\_c(t)$

mode04: $\forall t \cdot \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now) \land md\_c(t) = MODE\_F \Rightarrow g\_c(t) = zero$

mode03: $\forall t \cdot \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now) \land md\_c(t) = MODE\_G \Rightarrow p\_c(t) = g\_c(t)$

glue01: $p = p\_c(now)$

glue02: $f = f\_c(now)$

glue03: $g = g\_c(now)$

glue04: $md = md\_c(now)$

glue05: $active = \mathrm{TRUE} \Rightarrow \mathrm{leq}(active\_t, now)$

corridor01: $\forall t \cdot \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now) \Rightarrow \mathrm{leq}(p\_c(t), M)$

∀ t ∈ [0,now], p_c(t) ≤ M

corridor02: $active = \mathrm{TRUE} \Rightarrow$
$(\forall t \cdot \mathrm{leq}(active\_t, t) \land \mathrm{leq}(t, now) \Rightarrow \mathrm{leq}(m, p\_c(t)))$

active ⇒ ∀ t ∈ [active_t,now], m ≤ p_c(t)

mode05: $\forall t \cdot \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now) \land md\_c(t) = MODE\_G \Rightarrow f\_c(t) = zero$

THM_01: ⟨theorem⟩ $\forall t \cdot \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now) \Rightarrow md\_c(t) = MODE\_F \lor md\_c(t) = MODE\_G \lor md\_c(t) = MODE\_R$

glue06: $active = \mathrm{FALSE} \Rightarrow$
$(\forall t \cdot \mathrm{leq}(zero, t) \land \mathrm{leq}(t, now) \Rightarrow md\_c(t) = MODE\_F)$

¬active ⇒ ∀ t ∈ [0,now], md_c(t) = MODE_F

THM_02: ⟨theorem⟩ $\mathrm{leq}(now, now)$

now ≤ now

EVENTS

Initialisation

begin

act1: $active := \mathrm{FALSE}$
act4: $active\_t :\in REAL\_POS$

act3: $md := MODE\_F$
act2: $md\_c := \{zero \mapsto MODE\_F\}$
act6: $now := zero$
act7: $p\_c := \{zero \mapsto zero\}$
act8: $f\_c := \{zero \mapsto zero\}$
act9: $g\_c := \{zero \mapsto zero\}$

end

Event ENV_starting_f ⟨ordinary⟩ $\widehat{=}$
refines ENV_starting_f

any

dt
new_f_c

where

grd1: $active = \text{FALSE}$
grd2: $md\_c(now) = MODE\_F$
grd3: $\text{smr}(zero, dt)$
dt > 0
THM_2: ⟨theorem⟩ $\text{leq}(now, now \text{ plus } dt)$
now ≤ now + dt
THM_3: ⟨theorem⟩ $\text{leq}(zero, now \text{ plus } dt)$
0 ≤ now + dt
grd4: $new\_f\_c \in \{t | \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt)\} \rightarrow REAL\_POS$
new_f_c ∈ [now,now+dt] → R+
grd5: $f\_c(now) = new\_f\_c(now)$
grd6: $\forall t1, t2 \cdot t1 \in \text{dom}(new\_f\_c) \wedge t2 \in \text{dom}(new\_f\_c) \wedge \text{leq}(t1, t2)$
$\Rightarrow \text{leq}(new\_f\_c(t1), new\_f\_c(t2))$
∀ t1,t2 ∈ [now,now+dt], t1 ≤ t2 ⇒ new_f_c(t1) ≤ new_f_c(t2)
grd7: $\text{leq}(new\_f\_c(now \text{ plus } dt), M)$
THM_1: ⟨theorem⟩ $g\_c(now) = zero$

with

new_f: $new\_f = new\_f\_c(now \text{ plus } dt)$

then

act1: $now := now \text{ plus } dt$
act2: $p\_c := p\_c \Leftarrow new\_f\_c$
act3: $f\_c := f\_c \Leftarrow new\_f\_c$
act4: $g\_c := g\_c \Leftarrow (\lambda t \cdot \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt) | zero)$
act5: $md\_c := md\_c \Leftarrow (\lambda t \cdot \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt) | MODE\_F)$

end

Event CTRL_started ⟨ordinary⟩ $\widehat{=}$
refines CTRL_started

when

grd1: $active = \text{FALSE}$
grd2: $\text{leq}(m, p\_c(now))$
grd3: $\text{leq}(p\_c(now), M)$

then

act1: $active := \text{TRUE}$
act2: $active\_t := now$

end

Event ENV_evolution_f ⟨ordinary⟩ $\widehat{=}$
refines ENV_evolution_f

any

dt
new_f_c

where

grd1: $active = \mathrm{TRUE}$
grd2: $md\_c(now) = MODE\_F$
grd3: $\mathrm{smr}(zero, dt)$
dt > 0
THM_2: ⟨theorem⟩ $\mathrm{leq}(now, now \text{ plus } dt)$
now ≤ now + dt
THM_3: ⟨theorem⟩ $\mathrm{leq}(zero, now \text{ plus } dt)$
0 ≤ now + dt
grd8: $f\_c(now) \neq m$
grd9: $f\_c(now) \neq M$
grd4: $new\_f\_c \in \{t|\mathrm{leq}(now, t) \land \mathrm{leq}(t, now \text{ plus } dt)\} \to REAL\_POS$
new_f_c ∈ [now,now+dt] → R+
grd5: $f\_c(now) = new\_f\_c(now)$
grd6: $\forall t \cdot t \in \mathrm{dom}(new\_f\_c) \Rightarrow \mathrm{leq}(m, new\_f\_c(t))$
∀ t ∈ [now,now+dt], m ≤ new_f_c(t)
grda: $\forall t \cdot t \in \mathrm{dom}(new\_f\_c) \Rightarrow \mathrm{leq}(new\_f\_c(t), M)$
∀ t ∈ [now,now+dt], new_f_c(t) ≤ M
THM_1: ⟨theorem⟩ $g\_c(now) = zero$

with

new_f: $new\_f = new\_f\_c(now \text{ plus } dt)$

then

act1: $now := now \text{ plus } dt$
act2: $p\_c := p\_c \Leftarrow new\_f\_c$
act3: $f\_c := f\_c \Leftarrow new\_f\_c$
act4: $g\_c := g\_c \Leftarrow (\lambda t \cdot \mathrm{leq}(now, t) \land \mathrm{leq}(t, now \text{ plus } dt)|zero)$
act5: $md\_c := md\_c \Leftarrow (\lambda t \cdot \mathrm{leq}(now, t) \land \mathrm{leq}(t, now \text{ plus } dt)|MODE\_F)$

end

Event CTRL_limit_detected_f ⟨ordinary⟩ $\widehat{=}$
refines CTRL_limit_detected_f

when

grd2: $active = \mathrm{TRUE}$
grd1: $md\_c(now) = MODE\_F$
grd5: $f\_c(now) = m \lor f\_c(now) = M$
THM_1: ⟨theorem⟩ $g\_c(now) = zero$

then

act1: $md := MODE\_R$
act2: $md\_c(now) := MODE\_R$

end

Event ENV_evolution_fg ⟨ordinary⟩ $\widehat{=}$
refines ENV_evolution_fg

any

dt
new_f_c
new_g_c

where

grd1: $active = \mathrm{TRUE}$
grd2: $md\_c(now) = MODE\_R$
grd3: $\mathrm{smr}(zero, dt)$
dt > 0
THM_2: ⟨theorem⟩ $\mathrm{leq}(now, now \text{ plus } dt)$
now ≤ now + dt
THM_3: ⟨theorem⟩ $\mathrm{leq}(zero, now \text{ plus } dt)$
0 ≤ now + dt
grd4: $new\_f\_c \in \{t|\mathrm{leq}(now, t) \land \mathrm{leq}(t, now \text{ plus } dt)\} \to REAL\_POS$
new_f_c ∈ [now,now+dt] → R+

grd5: $f\_c(now) = new\_f\_c(now)$
grd7: $new\_g\_c \in \{t | \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt)\} \to REAL\_POS$
new_g_c ∈ [now,now+dt] → R+
grd8: $g\_c(now) = new\_g\_c(now)$
grd9: $\forall t1, t2 \cdot t1 \in \text{dom}(new\_f\_c) \wedge t2 \in \text{dom}(new\_f\_c) \wedge \text{leq}(t1, t2)$
$\Rightarrow \text{leq}(new\_f\_c(t2), new\_f\_c(t1))$
∀ t1,t2 ∈ [now,now+dt], t1 ≤ t2 ⇒ new_f_c(t2) ≤ new_f_c(t1)
grdb: $\forall t1, t2 \cdot t1 \in \text{dom}(new\_g\_c) \wedge t2 \in \text{dom}(new\_g\_c) \wedge \text{leq}(t1, t2)$
$\Rightarrow \text{leq}(new\_g\_c(t1), new\_g\_c(t2))$
∀ t1,t2 ∈ [now,now+dt], t1 ≤ t2 ⇒ new_g_c(t1) ≤ new_g_c(t2)
grdc: $\text{leq}(new\_g\_c(now \text{ plus } dt), M)$
grd6: $\forall t \cdot \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt) \Rightarrow \text{leq}(m, new\_f\_c(t) \text{ plus } new\_g\_c(t))$
∀ t ∈ [now,now+dt], m ≤ new_f_c(t) + new_g_c(t)
grda: $\forall t \cdot \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt) \Rightarrow \text{leq}(new\_f\_c(t) \text{ plus } new\_g\_c(t), M)$
∀ t ∈ [now,now+dt], new_f_c(t) + new_g_c(t) ≤ M

with

new_f: $new\_f = new\_f\_c(now \text{ plus } dt)$
new_g: $new\_g = new\_g\_c(now \text{ plus } dt)$

then

act1: $now := now \text{ plus } dt$
act3: $f\_c := f\_c \Leftarrow new\_f\_c$
act4: $g\_c := g\_c \Leftarrow new\_g\_c$
act2: $p\_c := p\_c \Leftarrow (\lambda t \cdot \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt) | new\_f\_c(t) \text{ plus } new\_g\_c(t))$
act5: $md\_c := md\_c \Leftarrow (\lambda t \cdot \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt) | MODE\_R)$

end

Event CTRL_repaired_g ⟨ordinary⟩ $\widehat{=}$
refines CTRL_repaired_g

when

grd2: $active = \text{TRUE}$
grd1: $md\_c(now) = MODE\_R$
grd3: $\text{leq}(m, g\_c(now))$
m ≤ g_c(now)
grd4: $\text{leq}(g\_c(now), M)$
g_c(now) ≤ M
grd5: $f\_c(now) = zero$
f_c(now) = 0

then

act1: $md := MODE\_G$
act2: $md\_c(now) := MODE\_G$

end

Event ENV_evolution_g ⟨ordinary⟩ $\widehat{=}$
refines ENV_evolution_g

any

dt
new_g_c

where

grd1: $active = \text{TRUE}$
grd2: $md\_c(now) = MODE\_G$
grd3: $\text{smr}(zero, dt)$
dt > 0
THM_2: ⟨theorem⟩ $\text{leq}(now, now \text{ plus } dt)$
now ≤ now + dt
THM_3: ⟨theorem⟩ $\text{leq}(zero, now \text{ plus } dt)$
0 ≤ now + dt

THM_4: ⟨theorem⟩ $\text{leq}(now \text{ plus } dt, now \text{ plus } dt)$

now + dt ≤ now + dt

grd4: $new\_g\_c \in \{t|\text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt)\} \rightarrow REAL\_POS$

new_g_c ∈ [now,now+dt] → R+

grd5: $g\_c(now) = new\_g\_c(now)$

grd6: $\forall t \cdot t \in \text{dom}(new\_g\_c) \Rightarrow \text{leq}(m, new\_g\_c(t))$

∀ t ∈ [now,now+dt], m ≤ new_g_c(t)

grda: $\forall t \cdot t \in \text{dom}(new\_g\_c) \Rightarrow \text{leq}(new\_g\_c(t), M)$

∀ t ∈ [now,now+dt], new_g_c(t) ≤ M

THM_1: ⟨theorem⟩ $f\_c(now) = zero$

with

new_g: $new\_g = new\_g\_c(now \text{ plus } dt)$

then

act1: $now := now \text{ plus } dt$

act2: $p\_c := p\_c \Leftarrow new\_g\_c$

act4: $f\_c := f\_c \Leftarrow (\lambda t \cdot \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt)|zero)$

act3: $g\_c := g\_c \Leftarrow new\_g\_c$

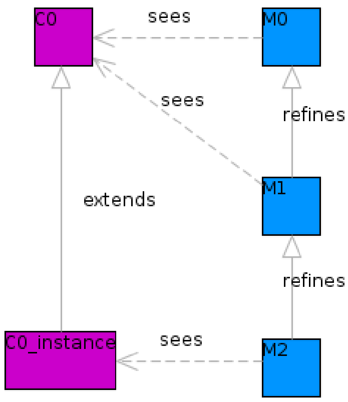
act5: $md\_c := md\_c \Leftarrow (\lambda t \cdot \text{leq}(now, t) \wedge \text{leq}(t, now \text{ plus } dt)|MODE\_G)$

end

END

E Generalization

Components:



The models are also available at: `http://babin.perso.enseeiht.fr/r/thesis/`

CONTEXT C0

ProB configuration:

- MAXINT = 6
- MAX_INITIALISATIONS = 1
- SYMBOLIC = TRUE
- TIME_OUT = 600000

ProB annotations:

Systems_states, *system_of*, *valuation_of*, *variables_of*, *fvar_of*, *varval_of* and *HorizontalInvs* are "symbolic"

SETS

Variables

ValueElements values that the variables can take

CONSTANTS

Valuations

VariablesSets each system has a set of variables

Systems

Systems_states

system_of system of a system state

valuation_of valuation of a system state

variables_of variables of a system state

fvar_of variant function of a system state

varval_of variant value of a system state

HorizontalInvs

AXIOMS

set1: $\mathrm{finite}(Variables)$

set2: $\mathrm{finite}(ValueElements)$

type1: $Valuations \subseteq Variables \rightarrow\!\!\!\!\!+ \mathbb{P}(ValueElements)$

type2: $VariablesSets \subseteq \mathbb{P}(Variables)$

prop1: $VariablesSets \neq \varnothing$

prop2: $\forall v1, v2 \cdot (v1 \in VariablesSets \land v2 \in VariablesSets \land v1 \neq v2)$
$\Rightarrow v1 \cap v2 = \varnothing$

systems do not share variables

type3: $Systems \subseteq VariablesSets \times (Valuations \rightarrow\!\!\!\!\!+ \mathbb{N})$

prop3a: $\mathrm{finite}(Systems)$

prop3b: $Systems \neq \varnothing$

prop4: $\forall vars, f\_var \cdot (vars \mapsto f\_var) \in Systems \Rightarrow$
$(\forall val \cdot val \in Valuations \Rightarrow$
$(val \in \mathrm{dom}(f\_var) \Leftrightarrow \mathrm{dom}(val) = vars))$

the variant function depends only on valuations whose domain is the system variables; and all valuations of the system variables are in the domain of the variant function

prop11: $\forall vars, f\_var \cdot (vars \mapsto f\_var) \in Systems \Rightarrow$
$\mathrm{dom}(f\_var) = vars \rightarrow \mathbb{P}(ValueElements)$

type4: $Systems\_states \subseteq Systems \times Valuations$

prop5: $Systems\_states \neq \varnothing$

prop12: $Systems\_states = (\bigcup sys \cdot sys \in Systems | \{sys\} \times (\mathrm{prj}_1(sys) \rightarrow \mathbb{P}(ValueElements)))$

prop6: $\mathrm{dom}(Systems\_states) = Systems$

prop7: $\forall sys\_st \cdot sys\_st \in Systems\_states \Rightarrow$
$\mathrm{dom}(\mathrm{prj}_2(sys\_st)) = \mathrm{prj}_1(\mathrm{prj}_1(sys\_st))$

the valuation depends on all system variables, and only those fun1: $system\_of = (\lambda sys\_st \cdot sys\_st \in Systems\_states | \mathrm{prj}_1(sys\_st))$

type5: ⟨theorem⟩ $\mathrm{dom}(system\_of) = Systems\_states$

fun2: $valuation\_of = (\lambda sys\_st \cdot sys\_st \in Systems\_states | \mathrm{prj}_2(sys\_st))$

type6: ⟨theorem⟩ $\mathrm{dom}(valuation\_of) = Systems\_states$

fun3: $variables\_of = (\lambda sys\_st \cdot sys\_st \in Systems\_states | \mathrm{prj}_1(\mathrm{prj}_1(sys\_st)))$

type7: ⟨theorem⟩ $\mathrm{dom}(variables\_of) = Systems\_states$

fun4: $fvar\_of = (\lambda sys\_st \cdot sys\_st \in Systems\_states | \mathrm{prj}_2(\mathrm{prj}_1(sys\_st)))$

type8: ⟨theorem⟩ $\mathrm{dom}(fvar\_of) = Systems\_states$

prop9: $\forall sys \cdot sys \in Systems \Rightarrow$
$\mathrm{ran}(\{sys\} \lhd Systems\_states) = \mathrm{dom}(\mathrm{prj}_2(sys))$

type11: ⟨theorem⟩ $\forall sys\_st \cdot sys\_st \in Systems\_states \Rightarrow \mathrm{prj}_2(sys\_st) \in \mathrm{dom}(fvar\_of(sys\_st))$

fun5: $varval\_of = (\lambda sys\_st \cdot sys\_st \in Systems\_states | fvar\_of(sys\_st)(\mathrm{prj}_2(sys\_st)))$

type9: ⟨theorem⟩ $\mathrm{dom}(varval\_of) = Systems\_states$

type10: $HorizontalInvs \in (Systems \times Systems) \mapsto ((Systems\_states \times Systems\_states) \mapsto \mathrm{BOOL})$

prop8: $\forall s1, s2, sst1, sst2, b \cdot ((s1 \mapsto s2) \mapsto \{(sst1 \mapsto sst2) \mapsto b\} \in HorizontalInvs)$
$\Rightarrow (s1 = system\_of(sst1)$
$\land s2 = system\_of(sst2))$

prop10: $\forall s1, s2 \cdot (s1 \mapsto s2) \in \mathrm{dom}(HorizontalInvs) \Rightarrow$
$\mathrm{dom}(HorizontalInvs(s1 \mapsto s2)) = (\{s1\} \times (\mathrm{prj}_1(s1) \rightarrow \mathbb{P}(ValueElements)))$
$\times (\{s2\} \times (\mathrm{prj}_1(s2) \rightarrow \mathbb{P}(ValueElements)))$

END

MACHINE M0
SEES C0
VARIABLES
    available_systems all the healthy systems
    current_system
INVARIANTS
    inv1: $available\_systems \subseteq Systems$
    inv2: $current\_system \in Systems$
EVENTS
Initialisation
  begin
    act1: $available\_systems := Systems$
    act2: $current\_system :\in Systems$
  end
Event failure ⟨ordinary⟩ $\widehat{=}$
  any
    system
  where
    grd1: $system \in available\_systems$
  then
    act1: $available\_systems := available\_systems \setminus \{system\}$
  end
Event treat_failure ⟨ordinary⟩ $\widehat{=}$
  any
    next_system
  where
    grd1: $next\_system \in available\_systems$
    grd2: $current\_system \notin available\_systems$
  then
    act1: $current\_system := next\_system$
  end
Event complete_failure ⟨ordinary⟩ $\widehat{=}$
  when
    grd1: $available\_systems = \varnothing$
  then
    skip
  end
END

MACHINE M1
REFINES M0
SEES C0
VARIABLES

available_systems all the healthy systems

available_systems_states

current_system

current_system_state

INVARIANTS

type1: $available\_systems\_states \subseteq Systems\_states$

type2: $current\_system\_state \in Systems\_states$

glue1: $available\_systems = \mathrm{dom}(available\_systems\_states)$

glue2: $current\_system = system\_of(current\_system\_state)$

VARIANT

$varval\_of(current\_system\_state)$ variant function of the current system, evaluated on the current values of the variables of the system

EVENTS

Initialisation

begin

act1: $available\_systems, available\_systems\_states :|$
$available\_systems\_states' = Systems\_states$
$\wedge\, available\_systems' = \mathrm{dom}(available\_systems\_states')$
$\wedge\, available\_systems' = Systems$

act2: $current\_system, current\_system\_state :|$
$current\_system\_state' \in Systems\_states$
$\wedge\, current\_system' = system\_of(current\_system\_state')$

end

Event failure ⟨ordinary⟩ $\widehat{=}$
extends failure

any

system

where

grd1: $system \in available\_systems$

then

act1: $available\_systems := available\_systems \setminus \{system\}$

act2: $available\_systems\_states := \{system\} \lhd available\_systems\_states$

end

Event treat_failure_with_state_repair ⟨ordinary⟩ $\widehat{=}$
refines treat_failure

any

new_variables
new_variant
new_valuation
h_inv

where

grd1: $current\_system \notin available\_systems$

grd2: $new\_variables \in VariablesSets$

grd3: $new\_variant \in Valuations \nrightarrow \mathbb{N}$

grd4: $new\_valuation \in Valuations$

grd5: $(new\_variables \mapsto new\_variant) \mapsto new\_valuation \in available\_systems\_states$

grd6: $new\_variables \neq variables\_of(current\_system\_state)$
different system
grd7: $new\_variant(new\_valuation) = varval\_of(current\_system\_state)$
same variant
grd10: $current\_system \mapsto (new\_variables \mapsto new\_variant) \in \mathrm{dom}(HorizontalInvs)$

grd8: $h\_inv = HorizontalInvs(current\_system \mapsto (new\_variables \mapsto new\_variant))$

grd9: $h\_\mathrm{inv}(current\_system\_state \mapsto ((new\_variables \mapsto new\_variant) \mapsto new\_valuation)) = \mathrm{TRUE}$

with
next_system: $next\_system = new\_variables \mapsto new\_variant$

then
act1: $current\_system := new\_variables \mapsto new\_variant$
act2: $current\_system\_state := (new\_variables \mapsto new\_variant) \mapsto new\_valuation$

end

Event complete_failure ⟨ordinary⟩ $\widehat{=}$
extends complete_failure
when
grd1: $available\_systems = \varnothing$
then
skip
end

Event progress ⟨convergent⟩ $\widehat{=}$
any
new_valuation
where
grd1: $current\_system \in available\_systems$
grd2: $new\_valuation \in Valuations$
grd3: $\mathrm{dom}(new\_valuation) = \mathrm{dom}(valuation\_of(current\_system\_state))$
same system variables
grd4: $fvar\_of(current\_system\_state)(new\_valuation) < varval\_of(current\_system\_state)$
the value of the variant decreases
then
act1: $current\_system\_state := system\_of(current\_system\_state) \mapsto new\_valuation$

end

END

CONTEXT C0_instance

ProB command (after export to ProB Classic):

```
probcli C0_instance_ctx.eventb -init -disable-timeout \
        -p MAXINT 6 -p MAX_INITIALISATIONS 1 -p SYMBOLIC TRUE
```

– with 1 product:
  execution: 2 to 2.5 sec ; peak memory usage: 157 MB
– with 2 products:
  execution: 1.5 to 2.5 sec ; peak memory usage: 158 MB
– with 3 products:
  execution: 5 to 6 sec ; peak memory usage: 244 MB
– with 4 products:
  execution: 47 sec ; peak memory usage: 6.83 GB
– with 5 products:
  execution: ?? ; peak memory usage: > 400 GB

EXTENDS C0

CONSTANTS

C1
C2a
C2b
Prod1
Prod2
Prod3
Prod4
Sys1
Sys2

AXIOMS

axm1: $\mathrm{partition}(Variables, \{C1\}, \{C2a\}, \{C2b\})$
carts

axm2: $\mathrm{partition}(ValueElements, \{Prod1\}, \{Prod2\}, \{Prod3\}, \{Prod4\})$
products

axm3: $Valuations = (\{C1\} \rightarrow \mathbb{P}(ValueElements)) \cup (\{C2a, C2b\} \rightarrow \mathbb{P}(ValueElements))$

axm4: $VariablesSets = \{\{C1\}, \{C2a, C2b\}\}$

axm5: $Sys1 = \{C1\} \mapsto (\lambda val \cdot val \in \{C1\} \rightarrow \mathbb{P}(ValueElements) \mid \mathrm{card}(ValueElements) - \mathrm{card}(val(C1)))$

axm6: $Sys2 = \{C2a, C2b\} \mapsto (\lambda val \cdot val \in \{C2a, C2b\} \rightarrow \mathbb{P}(ValueElements) \mid \mathrm{card}(ValueElements) - \mathrm{card}(val(C2a) \cup val(C2b)))$

axm7: $Systems = \{Sys1, Sys2\}$

axm8: $Systems\_states = (\{Sys1\} \times (\{C1\} \rightarrow \mathbb{P}(ValueElements))) \cup (\{Sys2\} \times (\{C2a, C2b\} \rightarrow \mathbb{P}(ValueElements)))$

axm9: $HorizontalInvs = \{ (Sys1 \mapsto Sys2) \mapsto (\lambda(sst1 \mapsto sst2) \cdot sst1 \in \{Sys1\} \times (\{C1\} \rightarrow \mathbb{P}(ValueElements)) \land sst2 \in \{Sys2\} \times (\{C2a, C2b\} \rightarrow \mathbb{P}(ValueElements)) \mid \mathrm{bool}(valuation\_of(sst1)(C1) = valuation\_of(sst2)(C2a) \cup valuation\_of(sst2)(C2b)))\}$

END

MACHINE M2
REFINES M1
SEES C0_instance
VARIABLES

available_systems all the healthy systems
available_systems_states
current_system
current_system_state
sys1_cart cart (in Sys1)
sys2_cart1 cart #1 (in Sys2)
sys2_cart2 cart #2 (in Sys2)

INVARIANTS

type1: $sys1\_cart \in \mathbb{P}(ValueElements)$
type2: $sys2\_cart1 \in \mathbb{P}(ValueElements)$
type3: $sys2\_cart2 \in \mathbb{P}(ValueElements)$
glue1: $system\_of(current\_system\_state) = Sys1 \Rightarrow$
$valuation\_of(current\_system\_state)(C1) = sys1\_cart$
glue2: $system\_of(current\_system\_state) = Sys2 \Rightarrow$
$valuation\_of(current\_system\_state)(C2a) = sys2\_cart1$
$\wedge\ valuation\_of(current\_system\_state)(C2b) = sys2\_cart2$
thm1: ⟨theorem⟩ $current\_system = Sys1 \Rightarrow$
$\{C1\} = \mathrm{dom}(valuation\_of(current\_system\_state))$
thm2: ⟨theorem⟩ $current\_system = Sys2 \Rightarrow$
$\{C2a, C2b\} = \mathrm{dom}(valuation\_of(current\_system\_state))$

EVENTS
Initialisation

begin

act1: $available\_systems := Systems$
act2: $available\_systems\_states := Systems\_states$
act3: $current\_system := Sys1$
act4: $current\_system\_state := Sys1 \mapsto \{C1 \mapsto \varnothing\}$
act5: $sys1\_cart := \varnothing$
act6: $sys2\_cart1 := \varnothing$
act7: $sys2\_cart2 := \varnothing$

end

Event failure_sys1 ⟨ordinary⟩ $\widehat{=}$
refines failure

when

grd1: $Sys1 \in available\_systems$

with

system: $system = Sys1$

then

act1: $available\_systems := available\_systems \setminus \{Sys1\}$
act2: $available\_systems\_states := \{Sys1\} \lhd available\_systems\_states$

end

Event failure_sys2 ⟨ordinary⟩ $\widehat{=}$
refines failure

when

grd1: $Sys2 \in available\_systems$

with

system: $system = Sys2$

then

act1: $available\_systems := available\_systems \setminus \{Sys2\}$

act2: $available\_systems\_states := \{Sys2\} \lhd available\_systems\_states$

end

Event treat_failure_with_state_repair__sys1_to_sys2 ⟨ordinary⟩ $\widehat{=}$

refines treat_failure_with_state_repair

any

new_sys2_cart1

new_sys2_cart2

where

grd1: $new\_sys2\_cart1 \in \mathbb{P}(ValueElements)$

grd2: $new\_sys2\_cart2 \in \mathbb{P}(ValueElements)$

grd3: $current\_system = Sys1$

grd4: $Sys1 \notin available\_systems$

grd5: $Sys2 \in available\_systems$

grd6: $sys1\_cart = new\_sys2\_cart1 \cup new\_sys2\_cart2$

grd7: $Sys2 \mapsto \{C2a \mapsto new\_sys2\_cart1, C2b \mapsto new\_sys2\_cart2\} \in available\_systems\_states$

with

new_variables: $new\_variables = \mathrm{prj}_1(Sys2)$

new_variant: $new\_variant = \mathrm{prj}_2(Sys2)$

new_valuation: $new\_valuation = \{C2a \mapsto new\_sys2\_cart1, C2b \mapsto new\_sys2\_cart2\}$

h_inv: $h\_inv = HorizontalInvs(Sys1 \mapsto Sys2)$

then

act1: $current\_system := Sys2$

act2: $current\_system\_state := Sys2 \mapsto \{C2a \mapsto new\_sys2\_cart1, C2b \mapsto new\_sys2\_cart2\}$

act3: $sys2\_cart1 := new\_sys2\_cart1$

act4: $sys2\_cart2 := new\_sys2\_cart2$

end

Event complete_failure ⟨ordinary⟩ $\widehat{=}$

extends complete_failure

when

grd1: $available\_systems = \varnothing$

then

skip

end

Event progress_sys1 ⟨convergent⟩ $\widehat{=}$

refines progress

any

new_prod

where

grd1: $current\_system = Sys1$

grd2: $Sys1 \in available\_systems$

grd3: $new\_prod \in ValueElements$

grd4: $new\_prod \notin sys1\_cart$

with

new_valuation: $new\_valuation = \{C1 \mapsto (sys1\_cart \cup \{new\_prod\})\}$

then

act1: $sys1\_cart := sys1\_cart \cup \{new\_prod\}$

act2: $current\_system\_state := Sys1 \mapsto \{C1 \mapsto (sys1\_cart \cup \{new\_prod\})\}$

end

Event progress_sys2_c1 ⟨convergent⟩ $\widehat{=}$

refines progress

any

new_prod

where

grd1: $current\_system = Sys2$

grd2: $Sys2 \in available\_systems$

grd3: $new\_prod \in ValueElements$

grd4: $new\_prod \notin sys2\_cart1$

grd5: $new\_prod \notin sys2\_cart2$

with

new_valuation: $new\_valuation = \{C2a \mapsto (sys2\_cart1 \cup \{new\_prod\}), C2b \mapsto sys2\_cart2\}$

then

act1: $sys2\_cart1 := sys2\_cart1 \cup \{new\_prod\}$

act2: $current\_system\_state := Sys2 \mapsto \{C2a \mapsto (sys2\_cart1 \cup \{new\_prod\}), C2b \mapsto sys2\_cart2\}$

end

Event progress_sys2_c2 ⟨convergent⟩ $\widehat{=}$

refines progress

any

new_prod

where

grd1: $current\_system = Sys2$

grd2: $Sys2 \in available\_systems$

grd3: $new\_prod \in ValueElements$

grd4: $new\_prod \notin sys2\_cart1$

grd5: $new\_prod \notin sys2\_cart2$

with

new_valuation: $new\_valuation = \{C2a \mapsto sys2\_cart1, C2b \mapsto (sys2\_cart2 \cup \{new\_prod\})\}$

then

act1: $sys2\_cart2 := sys2\_cart2 \cup \{new\_prod\}$

act2: $current\_system\_state := Sys2 \mapsto \{C2a \mapsto sys2\_cart1, C2b \mapsto (sys2\_cart2 \cup \{new\_prod\})\}$

end

END

Bibliography

[AA09] Idir Aït-Sadoune and Yamine Aït-Ameur. "A Proof Based Approach for Modelling and Verifying Web Services Compositions". In: *14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009), Potsdam, Germany.* June 2009, pp. 1–10. DOI: 10.1109/ICECCS.2009.48 (cit. on pp. 31, 32).

[AA10] Idir Aït-Sadoune and Yamine Aït-Ameur. "Stepwise Design of BPEL Web Services Compositions: An Event-B Refinement Based Approach". In: *8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2010), Montreal, Canada.* Ed. by Roger Lee, Olga Ormandjieva, Alain Abran, and Constantinos Constantinides. Vol. 296. Studies in Computational Intelligence. 8th ACIS Conference on Software Engineering Research, Management and Applications (SERA 2010), May 2010, Montréal, Canada. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 51–68. ISBN: 978-3-642-13273-5. DOI: 10.1007/978-3-642-13273-5_4. (Cit. on pp. 31, 32).

[AA13] Idir Aït-Sadoune and Yamine Aït-Ameur. "Stepwise Development of Formal Models for Web Services Compositions: Modelling and Property Verification". In: *Transactions on Large-Scale Data- and Knowledge-Centered Systems X.* Ed. by Abdelkader Hameurlain et al. Vol. 8220. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 1–33. ISBN: 978-3-642-41220-2. DOI: 10.1007/978-3-642-41221-9_1. (Cit. on pp. 31, 32).

[AA15] Idir Aït-Sadoune and Yamine Aït-Ameur. "Correct Software in Web Applications and Web Services". In: ed. by Bernhard Thalheim, Klaus-Dieter Schewe, Andreas Prinz, and Bruno Buchberger. Cham: Springer International Publishing, 2015. Chap. Formal Modelling and Verification of Transactional Web Service Composition: A Refinement and Proof Approach with Event-B, pp. 1–27. ISBN: 978-3-319-17112-8. DOI: 10.1007/978-3-319-17112-8_1. (Cit. on pp. 32, 33).

[Aal+09] Wil M. P. van der Aalst, Arjan J. Mooij, Christian Stahl, and Karsten Wolf. "Formal Methods for Web Services: 9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2009, Bertinoro, Italy, June 1-6, 2009, Advanced Lectures". In: ed. by Marco Bernardo, Luca Padovani, and

Gianluigi Zavattaro. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Chap. Service Interaction: Patterns, Formalization, and Analysis, pp. 42–88. ISBN: 978-3-642-01918-0. DOI: 10.1007/978-3-642-01918-0_2. (Cit. on p. 30).

[Abo+13] Faisal Abouzaid, Manuel Mazzara, John Mullins, and Nafees Qamar. "Towards a formal analysis of dynamic reconfiguration in WS-BPEL". In: *Intelligent Decision Technologies* 7.3 (June 2013), pp. 213–224. DOI: 10.3233/IDT-130164 (cit. on p. 31).

[Abr+09] Jean-Raymond Abrial et al. *Proposals for Mathematical Extensions for Event-B*. Tech. rep. 2009. URL: http://deploy-eprints.ecs.soton.ac.uk/216/ (cit. on p. 16).

[Abr+10] Jean-Raymond Abrial et al. "Rodin: an open toolset for modelling and reasoning in Event-B". In: *International Journal on Software Tools for Technology Transfer* 12.6 (2010), pp. 447–466. ISSN: 1433-2779. DOI: 10.1007/s10009-010-0145-y. (Cit. on pp. 16, 38, 80).

[Abr10] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. 1st. New York, NY, USA: Cambridge University Press, 2010. ISBN: 9780521895569 (cit. on pp. 10, 13, 14, 43, 80).

[Abr96] Jean-Raymond Abrial. *The B-Book: Assigning programs to meanings*. Cambridge University Press, 1996. ISBN: 978-0-521-02175-3. DOI: 10.1017/CBO9780511624162. (Cit. on pp. 10, 13, 14).

[ADM02] Eugene Asarin, Thao Dang, and Oded Maler. "The d/dt Tool for Verification of Hybrid Systems". In: *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings*. Ed. by Ed Brinksma and Kim Guldstrand Larsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 365–370. ISBN: 978-3-540-45657-5. DOI: 10.1007/3-540-45657-0_30. (Cit. on p. 36).

[AH07] Jean-Raymond Abrial and Stefan Hallerstede. "Refinement, decomposition, and instantiation of discrete models: Application to Event-B". In: *Fundamenta Informaticae* 77.1 (2007), pp. 1–28. ISSN: 1875-8681. URL: http://iospress.metapress.com/content/C74274T385T6R72R (cit. on p. 14).

[Akk+16] Ilge Akkaya, Patricia Derler, Shuhei Emoto, and Edward Ashford Lee. "Systems Engineering for Industrial Cyber-Physical Systems Using Aspects". In: *Proceedings of the IEEE* 104.5 (May 2016), pp. 997–1012. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2512265 (cit. on p. 34).

[Alu+95] Rajeev Alur et al. "The algorithmic analysis of hybrid systems". In: *Theoretical Computer Science* 138.1 (1995). Hybrid Systems, pp. 3–34. ISSN: 0304-3975. DOI: 10.1016/0304-3975(94)00202-T. (Cit. on p. 36).

[Alu11] Rajeev Alur. "Formal verification of hybrid systems". In: *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011*. Ed. by Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister. ACM, 2011, pp. 273–278. DOI: 10.1145/2038642.2038685. (Cit. on pp. 24, 79).

[An+15] Xin An et al. "Discrete Control-Based Design of Adaptive and Autonomic Computing Systems". In: *Distributed Computing and Internet Technology*. Ed. by Raja Natarajan, Gautam Barua, and Manas Ranjan Patra. Vol. 8956. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 93–113. ISBN: 978-3-319-14976-9. DOI: 10.1007/978-3-319-14977-6_6. (Cit. on p. 27).

[Arn88] André Arnold. "Mathematical problems in computation theory". In: ed. by Grażyna Mirkowska-Salwicka and Helena Rasiowa. Vol. 21. Banach Center Publications. PWN Polish Scientific Publishers, 1988. Chap. Transition systems and concurrent processes, pp. 9–20. ISBN: 978-8301079369 (cit. on p. 24).

[Bab+15] Guillaume Babin, Yamine Aït-Ameur, Shin Nakajima, and Marc Pantel. "Refinement and Proof Based Development of Systems Characterized by Continuous Functions". In: *Dependable Software Engineering: Theories, Tools, and Applications (SETTA)*. Ed. by Xuandong Li, Zhiming Liu, and Wang Yi. Vol. 9409. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 55–70. ISBN: 978-3-319-25941-3. DOI: 10.1007/978-3-319-25942-0_4. (Cit. on p. 95).

[Bab+16a] Guillaume Babin, Yamine Aït-Ameur, Neeraj Kumar Singh, and Marc Pantel. "A System Substitution Mechanism for Hybrid Systems in Event-B". In: *Formal Methods and Software Engineering: 18th International Conference on Formal Engineering Methods, ICFEM 2016, Tokyo, Japan, November 14-18, 2016, Proceedings*. Ed. by Kazuhiro Ogata, Mark Lawford, and Shaoying Liu. Vol. 10009. Lecture Notes in Computer Science. Springer International Publishing, Nov. 2016, pp. 106–121. ISBN: 978-3-319-47845-6. DOI: 10.1007/978-3-319-47846-3_8. (Cit. on p. 108).

[Bab+16b] Guillaume Babin, Yamine Aït-Ameur, Neeraj Kumar Singh, and Marc Pantel. "Handling Continuous Functions in Hybrid Systems Reconfigurations: A Formal Event-B Development". In: *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 5th International Conference, ABZ 2016, Linz, Austria, May 23-27, 2016, Proceedings*. Ed. by Michael Butler, Klaus-Dieter Schewe, Atif Mashkoor, and Miklos Biro. Springer International Publishing, 2016, pp. 290–296. ISBN: 978-3-319-33600-8. DOI: 10.1007/978-3-319-33600-8_23. (Cit. on p. 108).

[BAB16] Michael Butler, Jean-Raymond Abrial, and Richard Banach. "From Action Systems to Distributed Systems: The Refinement Approach". In: ed. by Luigia Petre and Emil Sekerinski. Computer and Information Science Series. Chapman and Hall/CRC, Apr. 2016. Chap. Modelling and Refining Hybrid Systems in Event-B and Rodin, pp. 29–42. ISBN: 978-1-49-870158-7. DOI: 10.1201/b20053-5. (Cit. on pp. 37, 38).

[Ban+11] Richard Banach, Huibiao Zhu, Wen Su, and Runlei Huang. "Formalising the Continuous/Discrete Modeling Step". In: *Proceedings 15th International Refinement Workshop, Refine 2011, Limerick, Ireland, 20th June 2011.* Ed. by John Derrick, Eerke A. Boiten, and Steve Reeves. Vol. 55. EPTCS. 2011, pp. 121–138. DOI: 10.4204/EPTCS.55.8. (Cit. on p. 38).

[Ban+12] Richard Banach, Huibiao Zhu, Wen Su, and Xiaofeng Wu. "ASM and Controller Synthesis". In: *Abstract State Machines, Alloy, B, VDM, and Z.* Ed. by John Derrick et al. Vol. 7316. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 51–64. ISBN: 978-3-642-30884-0. DOI: 10.1007/978-3-642-30885-7_4. (Cit. on p. 38).

[Ban+14] Richard Banach, Huibiao Zhu, Wen Su, and Xiaofeng Wu. "A Continuous ASM Modelling Approach to Pacemaker Sensing". In: *ACM Transactions on Software Engineering and Methodology* 24.1 (Oct. 2014), 2:1–2:40. ISSN: 1049-331X. DOI: 10.1145/2610375. (Cit. on p. 38).

[Ban+15] Richard Banach et al. "Core Hybrid Event-B I: Single Hybrid Event-B machines". In: *Science of Computer Programming* (2015). ISSN: 0167-6423. DOI: 10.1016/j.scico.2015.02.003. (Cit. on p. 38).

[Ban13] Richard Banach. "Pliant Modalities in Hybrid Event-B". In: *Theories of Programming and Formal Methods.* Ed. by Zhiming Liu, Jim Woodcock, and Huibiao Zhu. Vol. 8051. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 37–53. ISBN: 978-3-642-39697-7. DOI: 10.1007/978-3-642-39698-4_3. (Cit. on p. 38).

[Ban16a] Richard Banach. "Formal Refinement and Partitioning of a Fuel Pump System for Small Aircraft in Hybrid Event-B". In: *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE).* July 2016, pp. 65–72. DOI: 10.1109/TASE.2016.16 (cit. on p. 38).

[Ban16b] Richard Banach. "Hemodialysis Machine in Hybrid Event-B". In: *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 5th International Conference, ABZ 2016, Linz, Austria, May 23-27, 2016, Proceedings.* Ed. by Michael Butler, Klaus-Dieter Schewe, Atif Mashkoor, and Miklos Biro. Springer International Publishing, 2016, pp. 376–393. ISBN: 978-3-319-33600-8. DOI: 10.1007/978-3-319-33600-8_32. (Cit. on p. 38).

[BAP15] Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. "Formal Verification of Runtime Compensation of Web Service Compositions: A Refinement and Proof Based Proposal with Event-B". In: *2015 IEEE International Conference on Services Computing (SCC)*. June 2015, pp. 98–105. DOI: 10.1109/SCC.2015.23 (cit. on p. 76).

[BAP16a] Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. "A generic model for system substitution". In: *Trustworthy Cyber-Physical Systems Engineering*. Ed. by Alexander Romanovsky and Fuyuki Ishikawa. Computer and Information Science Series. Chapman and Hall/CRC, Sept. 2016. Chap. 4, pp. 75–103. ISBN: 9781498742450. URL: https://www.crcpress.com/Trustworthy-Cyber-Physical-Systems-Engineering/Romanovsky-Ishikawa/p/book/9781498742450 (cit. on p. 55).

[BAP16b] Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. "Correct Instantiation of a System Reconfiguration Pattern: A Proof and Refinement-Based Approach". In: *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*. Jan. 2016, pp. 31–38. DOI: 10.1109/HASE.2016.47 (cit. on p. 127).

[BAP17] Guillaume Babin, Yamine Aït-Ameur, and Marc Pantel. "Web Service Compensation at Runtime: Formal Modeling and Verification Using the Event-B Refinement and Proof Based Formal Method". In: *IEEE Transactions on Services Computing – Special Issue on Advances in Web Services Research* 10.1 (Jan. 2017), pp. 107–120. ISSN: 1939-1374. DOI: 10.1109/TSC.2016.2594782 (cit. on p. 76).

[BBG07] Maurice ter Beek, Antonio Bucchiarone, and Stefania Gnesi. "Web Service Composition Approaches: From Industrial Standards to Formal Methods". In: *Proceedings of the 2nd International Conference on Internet and Web Applications and Services (ICIW 2007)*. IEEE Computer Society Press, May 2007, pp. 15–21. DOI: 10.1109/ICIW.2007.71 (cit. on p. 30).

[BC04] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004, p. 470. ISBN: 3-540-20854-2. URL: https://www.springer.com/computer/swe/book/978-3-540-20854-9 (cit. on pp. 10, 37, 127).

[BF04] Michael Butler and Carla Ferreira. "An Operational Semantics for StAC, a Language for Modelling Long-Running Business Transactions". In: *Coordination Models and Languages: 6th International Conference, COORDINATION 2004 Pisa Italy, February 24-27, 2004 Proceedings*. Ed. by Rocco De Nicola, Gian-Luigi Ferrari, and Greg Meredith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 87–104. ISBN: 978-3-540-24634-3. DOI: 10.1007/978-3-540-24634-3_9. (Cit. on p. 30).

[BFN05] Michael Butler, Carla Ferreira, and Muan Yong Ng. "Precise Modelling of Compensating Business Transactions and its Application to BPEL". In: *Journal of Universal Computer Science* 11.5 (May 28, 2005), pp. 712–743. DOI: 10.3217/jucs-011-05-0712 (cit. on p. 31).

[Bha13] Anirban Bhattacharyya. "Formal Modelling and Analysis of Dynamic Reconfiguration of Dependable Systems". PhD thesis. Newcastle University School of Computing Science, Jan. 2013 (cit. on p. 27).

[BHF05] Michael Butler, Tony Hoare, and Carla Ferreira. "A Trace Semantics for Long-Running Transactions". In: *Communicating Sequential Processes. The First 25 Years: Symposium on the Occasion of 25 Years of CSP, London, UK, July 7-8, 2004. Revised Invited Papers*. Ed. by Ali E. Abdallah, Cliff B. Jones, and Jeff W. Sanders. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 133–150. ISBN: 978-3-540-32265-8. DOI: 10.1007/11423348_8. (Cit. on p. 30).

[BJ78] Dines Bjørner and Cliff B. Jones, eds. *The Vienna Development Method: The Meta-Language*. Vol. 61. Lecture Notes in Computer Science. Springer, 1978. ISBN: 3-540-08766-4 (cit. on p. 10).

[BLM15] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. "Coquelicot: A User-Friendly Library of Real Analysis for Coq". In: *Mathematics in Computer Science* 9.1 (2015), pp. 41–62. ISSN: 1661-8270. DOI: 10.1007/s11786-014-0181-1. (Cit. on pp. 37, 94).

[BM13] Michael Butler and Issam Maamria. "Practical Theory Extension in Event-B". In: *Theories of Programming and Formal Methods*. Ed. by Zhiming Liu, Jim Woodcock, and Huibiao Zhu. Vol. 8051. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 67–81. ISBN: 978-3-642-39697-7. DOI: 10.1007/978-3-642-39698-4_5. (Cit. on p. 16).

[Bol+14] Sylvie Boldo et al. "Trusting computations: A mechanized proof from partial differential equations to actual program". In: *Computers & Mathematics with Applications* 68.3 (2014), pp. 325–352. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2014.06.004. (Cit. on pp. 36, 38).

[BR05] Michael Butler and Shamim Ripon. "Executable Semantics for Compensating CSP". In: *Formal Techniques for Computer Systems and Business Processes: European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005. Proceedings*. Ed. by Mario Bravetti, Leïla Kloul, and Gianluigi Zavattaro. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 243–256. ISBN: 978-3-540-31903-0. DOI: 10.1007/11549970_18. (Cit. on p. 30).

[Bru+05] Roberto Bruni et al. "Comparing Two Approaches to Compensable Flow Composition". In: *CONCUR 2005 – Concurrency Theory: 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005. Proceedings*. Ed. by Martín Abadi and Luca de

Alfaro. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 383–397. ISBN: 978-3-540-31934-4. DOI: 10.1007/11539452_30. (Cit. on p. 30).

[BT08] Egon Börger and Bernhard Thalheim. "Modeling Workflows, Interaction Patterns, Web Services and Business Processes: The ASM-Based Approach". In: *Abstract State Machines, B and Z: First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings*. Ed. by Egon Börger, Michael Butler, Jonathan P. Bowen, and Paul Boca. Vol. 5238. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 24–38. ISBN: 978-3-540-87603-8. DOI: 10.1007/978-3-540-87603-8_3. (Cit. on p. 31).

[Bur+92] Jerry R. Burch et al. "Symbolic model checking: 10^{20} States and beyond". In: *Information and Computation* 98.2 (1992), pp. 142–170. ISSN: 0890-5401. DOI: http://dx.doi.org/10.1016/0890-5401(92)90017-A. (Cit. on p. 10).

[BV06] Bernard Berthomieu and François Vernadat. "Time Petri Nets Analysis with TINA". In: *Third International Conference on the Quantitative Evaluation of Systems (QEST'06)*. Sept. 2006, pp. 123–124. DOI: 10.1109/QEST.2006.56 (cit. on p. 10).

[BW10] Jeremy W. Bryans and Wei Wei. "Formal Analysis of BPMN Models Using Event-B". In: *Formal Methods for Industrial Critical Systems: 15th International Workshop, FMICS 2010, Antwerp, Belgium, September 20-21, 2010. Proceedings*. Ed. by Stefan Kowalewski and Marco Roveri. Vol. 6371. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 33–49. ISBN: 978-3-642-15898-8. DOI: 10.1007/978-3-642-15898-8_3. (Cit. on p. 31).

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Flow*: An Analyzer for Non-linear Hybrid Systems". In: *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Ed. by Natasha Sharygina and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–263. ISBN: 978-3-642-39799-8. DOI: 10.1007/978-3-642-39799-8_18. (Cit. on p. 36).

[CC77] Patrick Cousot and Radhia Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". In: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '77. Los Angeles, California: ACM, 1977, pp. 238–252. DOI: 10.1145/512950.512973. (Cit. on p. 94).

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. The MIT Press, Dec. 1999. ISBN: 9780262032704. URL: https://mitpress.mit.edu/books/model-checking (cit. on p. 36).

[Cou+05] Patrick Cousot et al. "The ASTRÉE Analyzer". In: *Programming Languages and Systems*. Ed. by Mooly Sagiv. Vol. 3444. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 21–30. ISBN: 978-3-540-25435-5. DOI: 10.1007/978-3-540-31987-0_3. (Cit. on p. 94).

[Dij97] Edsger W. Dijkstra. *A Discipline of Programming*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997. ISBN: 013215871X (cit. on pp. 10, 13).

[Egg+11] Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. "Improving SAT Modulo ODE for Hybrid Systems Analysis by Combining Different Enclosure Methods". In: *Software Engineering and Formal Methods: 9th International Conference, SEFM 2011, Montevideo, Uruguay, November 14-18, 2011. Proceedings*. Ed. by Gilles Barthe, Alberto Pardo, and Gerardo Schneider. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 172–187. ISBN: 978-3-642-24690-6. DOI: 10.1007/978-3-642-24690-6_13. (Cit. on p. 36).

[Ehr+10] Hartmut Ehrig et al. "Formal Analysis and Verification of Self-Healing Systems". In: *Fundamental Approaches to Software Engineering: 13th International Conference, FASE 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*. Ed. by David S. Rosenblum and Gabriele Taentzer. Vol. 6013. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 139–153. ISBN: 978-3-642-12029-9. DOI: 10.1007/978-3-642-12029-9_10. (Cit. on p. 31).

[EVD88] P. H. J. van Eijk, C. A. Vissers, and Michel Diaz, eds. *The Formal Description Technique LOTOS*. North Holland, Amsterdam, Dec. 1988. ISBN: 9780444872678 (cit. on p. 10).

[Fer04] Andrea Ferrara. "Web Services: A Process Algebra Approach". In: *Proceedings of the 2nd International Conference on Service Oriented Computing*. ICSOC '04. New York, NY, USA: ACM, 2004, pp. 242–251. ISBN: 1-58113-871-7. DOI: 10.1145/1035167.1035202. (Cit. on pp. 30, 31).

[FGT12] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. "A formal approach to adaptive software: continuous assurance of non-functional requirements". In: *Formal Aspects of Computing* 24.2 (2012), pp. 163–186. ISSN: 0934-5043. DOI: 10.1007/s00165-011-0207-2. (Cit. on p. 28).

[FM07] Jean-Christophe Filliâtre and Claude Marché. "The Why/Krakatoa/-Caduceus Platform for Deductive Program Verification". In: *Computer Aided Verification: 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007. Proceedings*. Ed. by Werner Damm and Holger Hermanns. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 173–177. ISBN: 978-3-540-73368-3. DOI: 10.1007/978-3-540-73368-3_21. (Cit. on p. 37).

[Fos+06] Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. "LTSA-WS: A Tool for Model-based Verification of Web Service Compositions and Choreography". In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06. Shanghai, China: ACM, 2006, pp. 771–774. ISBN: 1-59593-375-1. DOI: 10.1145/1134285.1134408. (Cit. on p. 30).

[Frä+07] Martin Fränzle et al. "Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure". In: *Journal on Satisfiability, Boolean Modeling and Computation* 1.3-4 (2007), pp. 209–236. URL: https://www.satassociation.org/jsat/index.php/jsat/article/view/16 (cit. on p. 36).

[Fre+11] Goran Frehse et al. "SpaceEx: Scalable Verification of Hybrid Systems". In: *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 379–395. ISBN: 978-3-642-22110-1. DOI: 10.1007/978-3-642-22110-1_30. (Cit. on p. 36).

[Fre08] Goran Frehse. "PHAVer: algorithmic verification of hybrid systems past HyTech". In: *International Journal on Software Tools for Technology Transfer* 10.3 (2008), pp. 263–279. ISSN: 1433-2787. DOI: 10.1007/s10009-007-0062-x. (Cit. on p. 36).

[Gar+13] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. "CADP 2011: a toolbox for the construction and analysis of distributed processes". In: *International Journal on Software Tools for Technology Transfer* 15.2 (2013), pp. 89–107. ISSN: 1433-2787. DOI: 10.1007/s10009-012-0244-z. (Cit. on p. 10).

[GKC13a] Sicun Gao, Soonho Kong, and Edmund M. Clarke. "dReal: An SMT Solver for Nonlinear Theories over the Reals". In: *Automated Deduction – CADE-24: 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*. Ed. by Maria Paola Bonacina. Springer Berlin Heidelberg, 2013, pp. 208–214. ISBN: 978-3-642-38574-2. DOI: 10.1007/978-3-642-38574-2_14. (Cit. on p. 36).

[GKC13b] Sicun Gao, Soonho Kong, and Edmund M. Clarke. "Satisfiability modulo ODEs". In: *2013 Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*. Oct. 2013, pp. 105–112. DOI: 10.1109/FMCAD.2013.6679398. (Cit. on p. 36).

[Gou01] Eric Goubault. “Static Analyses of the Precision of Floating-Point Operations”. In: *Static Analysis*. Ed. by Patrick Cousot. Vol. 2126. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 234–259. ISBN: 978-3-540-42314-0. DOI: 10.1007/3-540-47764-0_14. (Cit. on p. 94).

[HA11] Thái Sơn Hoàng and Jean-Raymond Abrial. “Reasoning about Liveness Properties in Event-B”. In: *Formal Methods and Software Engineering*. Ed. by Shengchao Qin and Zongyan Qiu. Vol. 6991. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 456–471. ISBN: 978-3-642-24558-9. DOI: 10.1007/978-3-642-24559-6_31. (Cit. on p. 15).

[Har87] David Harel. “Statecharts: a visual formalism for complex systems”. In: *Science of Computer Programming* 8.3 (1987), pp. 231–274. ISSN: 0167-6423. DOI: 10.1016/0167-6423(87)90035-9. (Cit. on pp. 10, 61).

[He+08] Yanxiang He, Liang Zhao, Zhao Wu, and Fei Li. “Formal Modeling of Transaction Behavior in WS-BPEL”. In: *Computer Science and Software Engineering, 2008 International Conference on*. Vol. 3. Dec. 2008, pp. 490–494. DOI: 10.1109/CSSE.2008.873 (cit. on p. 30).

[Hen00] Thomas A. Henzinger. “The Theory of Hybrid Automata”. In: *Verification of Digital and Hybrid Systems*. Ed. by M. Kemal Inan and Robert P. Kurshan. Vol. 170. NATO ASI Series. Springer Berlin Heidelberg, 2000, pp. 265–292. ISBN: 978-3-642-64052-0. DOI: 10.1007/978-3-642-59615-5_13. (Cit. on pp. 36, 79).

[HHW97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. “HyTech: A Model Checker for Hybrid Systems”. In: *International Journal on Software Tools for Technology Transfer* 1.1-2 (1997), pp. 110–122. ISSN: 1433-2779. DOI: 10.1007/s100090050008. (Cit. on p. 36).

[Hoa+17] Thai Son Hoang et al. “Theory Plug-in for Rodin 3.x”. In: *Computing Research Repository (CoRR)* abs/1701.08625 (2017). Event-B day, NII (National Institute of Informatics), 21 November 2016, Tokyo, Japan, pp. 1–9. URL: http://arxiv.org/abs/1701.08625 (cit. on p. 16).

[Hoa69] C. A. R. Hoare. “An Axiomatic Basis for Computer Programming”. In: *Communications of the ACM* 12.10 (Oct. 1969), pp. 576–580. ISSN: 0001-0782. DOI: 10.1145/363235.363259. (Cit. on p. 10).

[Hol04] Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, Sept. 2004. ISBN: 978-0321228628 (cit. on p. 10).

[HSS05] Sebastian Hinz, Karsten Schmidt, and Christian Stahl. “Transforming BPEL to Petri Nets”. In: *Business Process Management: 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005. Proceedings*. Ed. by Wil M. P. van der Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera. Berlin, Heidelberg: Springer

Berlin Heidelberg, 2005, pp. 220–235. ISBN: 978-3-540-31929-0. DOI: 10.1007/11538394_15. (Cit. on p. 30).

[IMN13] Daisuke Ishii, Guillaume Melquiond, and Shin Nakajima. "Inductive Verification of Hybrid Automata with Strongest Postcondition Calculus". In: *Integrated Formal Methods*. Ed. by Einar Broch Johnsen and Luigia Petre. Vol. 7940. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 139–153. ISBN: 978-3-642-38612-1. DOI: 10.1007/978-3-642-38613-8_10. (Cit. on p. 36).

[ISO02] ISO. *Information technology – Z formal specification notation – Syntax, type system and semantics*. Standard ISO/IEC 13568:2002. Geneva, CH: International Organization for Standardization, July 2002. URL: https://www.iso.org/standard/21573.html (cit. on p. 10).

[ISO89] ISO. *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*. Standard ISO 8807:1989. Geneva, CH: International Organization for Standardization, Feb. 1989. URL: https://www.iso.org/standard/16258.html (cit. on p. 10).

[Kon+15] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. "dReach: δ-Reachability Analysis for Hybrid Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*. Ed. by Christel Baier and Cesare Tinelli. Springer Berlin Heidelberg, 2015, pp. 200–205. ISBN: 978-3-662-46681-0. DOI: 10.1007/978-3-662-46681-0_15. (Cit. on p. 36).

[Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, June 2002. ISBN: 0-321-14306-X. URL: https://www.microsoft.com/en-us/research/publication/specifying-systems-the-tla-language-and-tools-for-hardware-and-software-engineers/ (cit. on p. 10).

[LB03] Michael Leuschel and Michael Butler. "ProB: A Model Checker for B". In: *FME 2003: Formal Methods*. Ed. by Keijiro Araki, Stefania Gnesi, and Dino Mandrioli. Vol. 2805. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 855–874. ISBN: 978-3-540-40828-4. DOI: 10.1007/978-3-540-45236-2_46. (Cit. on p. 10).

[LCR06] Rogério de Lemos, Paulo Asterio de Castro Guerra, and Cecília Mary Fischer Rubira. "A fault-tolerant architectural approach for dependable systems". In: *IEEE Software* 23.2 (Mar. 2006), pp. 80–87. ISSN: 0740-7459. DOI: 10.1109/MS.2006.35. (Cit. on p. 27).

[LDK11] Arnaud Lanoix, Julien Dormoy, and Olga Kouchnarenko. "Combining Proof and Model-checking to Validate Reconfigurable Architectures". In: *Electronic Notes in Theoretical Computer Science* 279.2 (2011). Proceedings of the 8th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA), pp. 43–57. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2011.11.011. (Cit. on p. 27).

[Lee14] Edward Ashford Lee. *Constructive Models of Discrete and Continuous Physical Phenomena*. Tech. rep. UCB/EECS-2014-15. EECS Department, University of California, Berkeley, Feb. 2014. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-15.html (cit. on p. 34).

[Lee15] Edward Ashford Lee. "The Past, Present and Future of Cyber-Physical Systems: A Focus on Models". In: *Sensors* 15.3 (2015), p. 4837. ISSN: 1424-8220. DOI: 10.3390/s150304837. (Cit. on p. 34).

[Lem+13] Rogério Lemos et al. "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap". In: *Software Engineering for Self-Adaptive Systems II*. Ed. by Rogério Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw. Vol. 7475. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 1–32. ISBN: 978-3-642-35812-8. DOI: 10.1007/978-3-642-35813-5_1. (Cit. on p. 27).

[LM07] Roberto Lucchi and Manuel Mazzara. "A pi-calculus based semantics for WS-BPEL". In: *The Journal of Logic and Algebraic Programming* 70.1 (2007), pp. 96–118. ISSN: 1567-8326. DOI: 10.1016/j.jlap.2006.05.007. (Cit. on p. 30).

[Loh+08] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. "Analyzing interacting WS-BPEL processes using flexible model generation". In: *Data & Knowledge Engineering* 64.1 (2008). Fourth International Conference on Business Process Management (BPM 2006)8th International Conference on Enterprise Information Systems (ICEIS' 2006), pp. 38–54. ISSN: 0169-023X. DOI: 10.1016/j.datak.2007.06.006. (Cit. on p. 30).

[LR14] Ilya Lopatkin and Alexander Romanovsky. "Rigorous Development of Fault-Tolerant Systems through Co-refinement". In: *Reliable Software Technologies – Ada-Europe 2014: 19th Ada-Europe International Conference on Reliable Software Technologies, Paris, France, June 23-27, 2014. Proceedings*. Ed. by Laurent George and Tullio Vardanega. Springer International Publishing, 2014, pp. 11–26. ISBN: 978-3-319-08311-7. DOI: 10.1007/978-3-319-08311-7_3. (Cit. on p. 27).

[LS14] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. 1.5. LeeSeshia.org, 2014. ISBN: 978-1-312-42740-2. URL: http://leeseshia.org/ (cit. on pp. 24, 34).

[LZ13] Ivan Lanese and Gianluigi Zavattaro. "Decidability Results for Dynamic Installation of Compensation Handlers". In: *Coordination Models and Languages: 15th International Conference, COORDINATION 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings*. Ed. by Rocco De Nicola and Christine Julien. Vol. 7890. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 136–150. ISBN: 978-3-642-38493-6. DOI: 10.1007/978-3-642-38493-6_10. (Cit. on p. 31).

[Mar07] Claude Marché. "Jessie: An Intermediate Language for Java and C Verification". In: *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification*. PLPV '07. Freiburg, Germany: ACM, 2007, pp. 1–2. ISBN: 978-1-59593-677-6. DOI: 10.1145/1292597.1292598. (Cit. on p. 37).

[MGZ14] Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. "Web Services Foundations". In: ed. by Athman Bouguettaya, Z. Quan Sheng, and Florian Daniel. New York, NY: Springer New York, 2014. Chap. Service-Oriented Programming with Jolie, pp. 81–107. ISBN: 978-1-4614-7518-7. DOI: 10.1007/978-1-4614-7518-7_4. (Cit. on p. 31).

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science 0302-9743. Springer Berlin Heidelberg, 1980. ISBN: 978-3-540-10235-9. DOI: 10.1007/3-540-10235-3 (cit. on pp. 10, 68).

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall International Series in Computer Science. Prentice-Hall, 1989. ISBN: 978-0131149847 (cit. on p. 68).

[MP09] Annapaola Marconi and Marco Pistore. "Formal Methods for Web Services: 9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2009, Bertinoro, Italy, June 1-6, 2009, Advanced Lectures". In: ed. by Marco Bernardo, Luca Padovani, and Gianluigi Zavattaro. Vol. 5569. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Chap. Synthesis and Composition of Web Services, pp. 89–157. ISBN: 978-3-642-01918-0. DOI: 10.1007/978-3-642-01918-0_3. (Cit. on p. 30).

[MPS14] Raffaela Mirandola, Pasqualina Potena, and Patrizia Scandurra. "Adaptation space exploration for service-oriented applications". In: *Science of Computer Programming* 80, Part B (2014), pp. 356–384. ISSN: 0167-6423. DOI: 10.1016/j.scico.2013.09.017. (Cit. on p. 28).

[Mul+10] Jean-Michel Muller et al. *Handbook of Floating-Point Arithmetic*. Birkhäuser, 2010. ISBN: 978-0-8176-4704-9. DOI: 10.1007/978-0-8176-4705-6. (Cit. on p. 94).

[Nak06] Shin Nakajima. "Model-Checking Behavioral Specification of BPEL Applications". In: *Electronic Notes in Theoretical Computer Science* 151.2 (2006). Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005), pp. 89–105. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2005.07.038. (Cit. on p. 30).

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Vol. 2283. Lecture Notes in Computer Science. Springer, 2002. ISBN: 3-540-43376-7. DOI: 10.1007/3-540-45949-9. (Cit. on pp. 10, 127).

[OAS07] OASIS. *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Apr. 2007. URL: http://bpel.xml.org/ (cit. on p. 29).

[OMG14] OMG. *Business Process Model and Notation (BPMN) Version 2.0.2*. Jan. 2014. URL: http://www.omg.org/spec/BPMN/2.0.2/ (cit. on p. 29).

[OMG15] OMG. *Unified Modeling Language (OMG UML), Version 2.5*. Mar. 2015. URL: http://www.omg.org/spec/UML/2.5/ (cit. on p. 61).

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. "PVS: A prototype verification system". In: *Automated Deduction—CADE-11: 11th International Conference on Automated Deduction Saratoga Springs, NY, USA, June 15–18, 1992 Proceedings*. Ed. by Deepak Kapur. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 748–752. ISBN: 978-3-540-47252-0. DOI: 10.1007/3-540-55602-8_217. (Cit. on p. 10).

[PC09] André Platzer and Edmund M. Clarke. "Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study". In: *FM 2009: Formal Methods: Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*. Ed. by Ana Cavalcanti and Dennis R. Dams. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 547–562. ISBN: 978-3-642-05089-3. DOI: 10.1007/978-3-642-05089-3_35. (Cit. on p. 37).

[PH05] Manish Parashar and Salim Hariri. "Autonomic Computing: An Overview". In: *Unconventional Programming Paradigms*. Ed. by Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel. Vol. 3566. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 257–269. ISBN: 978-3-540-27884-9. DOI: 10.1007/11527800_20. (Cit. on p. 27).

[PL10] Daniel Plagge and Michael Leuschel. “Seven at one stroke: LTL model checking for high-level specifications in B, Z, CSP, and more”. In: *International Journal on Software Tools for Technology Transfer* 12.1 (Feb. 2010), pp. 9–21. ISSN: 1433-2787. DOI: 10.1007/s10009-009-0132-3. (Cit. on p. 15).

[Pla08] André Platzer. “Differential Dynamic Logic for Hybrid Systems”. In: *Journal of Automated Reasoning* 41.2 (2008), pp. 143–189. ISSN: 1573-0670. DOI: 10.1007/s10817-008-9103-8. (Cit. on pp. 37, 38).

[PLB01] Noël de Palma, Philippe Laumay, and Luc Bellissard. “Ensuring Dynamic Reconfiguration Consistency”. In: *In 6th International Workshop on Component-Oriented Programming (WCOP 2001), ECOOP related Workshop*. 2001, pp. 18–24 (cit. on p. 27).

[Pot13] Pasqualina Potena. “Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff”. In: *Journal of Systems and Software* 86.3 (2013), pp. 624–648. ISSN: 0164-1212. DOI: 10.1016/j.jss.2012.10.929. (Cit. on p. 28).

[PQ09] André Platzer and Jan-David Quesel. “European Train Control System: A Case Study in Formal Verification”. In: *Formal Methods and Software Engineering: 11th International Conference on Formal Engineering Methods ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009. Proceedings*. Ed. by Karin Breitman and Ana Cavalcanti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 246–265. ISBN: 978-3-642-10373-5. DOI: 10.1007/978-3-642-10373-5_13. (Cit. on p. 37).

[PTL12] Inna Pereverzeva, Elena Troubitsyna, and Linas Laibinis. “Development of Fault Tolerant MAS with Cooperative Error Recovery by Refinement in Event-B”. In: *DS-Event-B 2012: Workshop on the experience of and advances in developing dependable systems in Event-B, in conjunction with ICFEM 2012 - Kyoto, Japan, November 13, 2012*. Ed. by Fuyuki Ishikawa and Alexander Romanovsky. 2012. URL: http://arxiv.org/abs/1210.7035 (cit. on p. 27).

[PTL13] Inna Pereverzeva, Elena Troubitsyna, and Linas Laibinis. “A refinement-based approach to developing critical multi-agent systems”. In: *International Journal of Critical Computer-Based Systems* 4.1 (Jan. 2013), pp. 69–91. DOI: 10.1504/IJCCBS.2013.053743. (Cit. on p. 27).

[Pto14] Claudius Ptolemaeus, ed. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. ISBN: 978-1-304-42106-7. URL: http://ptolemy.org/books/Systems (cit. on p. 35).

[Que+16] Jan-David Quesel et al. “How to model and prove hybrid systems with KeYmaera: a tutorial on safety”. In: *International Journal on Software Tools for Technology Transfer* 18.1 (2016), pp. 67–91. ISSN: 1433-2787. DOI: 10.1007/s10009-015-0367-0. (Cit. on pp. 37, 38).

[Rod+12] Rodrigo Rodrigues et al. "Automatic Reconfiguration for Large-Scale Reliable Storage Systems". In: *Dependable and Secure Computing, IEEE Transactions on* 9.2 (Mar. 2012), pp. 145–158. ISSN: 1545-5971. DOI: 10.1109/TDSC.2010.52. (Cit. on p. 27).

[SA14] Wen Su and Jean-Raymond Abrial. "Aircraft Landing Gear System: Approaches with Event-B to the Modeling of an Industrial System". In: *ABZ 2014: The Landing Gear Case Study: Case Study Track, Held at the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z, Toulouse, France, June 2-6, 2014. Proceedings.* Ed. by Frédéric Boniol, Virginie Wiels, Yamine Aït-Ameur, and Klaus-Dieter Schewe. Springer International Publishing, 2014, pp. 19–35. ISBN: 978-3-319-07512-9. DOI: 10.1007/978-3-319-07512-9_2. (Cit. on p. 37).

[SAZ14] Wen Su, Jean-Raymond Abrial, and Huibiao Zhu. "Formalizing hybrid systems with Event-B and the Rodin Platform". In: *Science of Computer Programming* 94, Part 2 (2014). Abstract State Machines, Alloy, B, VDM, and Z Selected and extended papers from ABZ 2012, pp. 164–202. ISSN: 0167-6423. DOI: 10.1016/j.scico.2014.04.015. (Cit. on pp. 37, 38, 80, 85, 102).

[SBS04] Gwen Salaün, Lucas Bordeaux, and Marco Schaerf. "Describing and reasoning on Web services using process algebra". In: *IEEE International Conference on Web Services (ICWS 2004)*. July 2004, pp. 43–50. ISBN: 0-7695-2167-3. DOI: 10.1109/ICWS.2004.1314722 (cit. on p. 30).

[Spi92] John Michael Spivey. *The Z Notation: A Reference Manual*. Second Edition. International Series in Computer Science. Prentice Hall, June 1992. ISBN: 978-0139785290. URL: http://spivey.oriel.ox.ac.uk/mike/zrm/ (cit. on p. 10).

[Tar+12] Anton Tarasyuk et al. "Formal Development and Assessment of a Reconfigurable On-board Satellite System". In: *Computer Safety, Reliability, and Security*. Ed. by Frank Ortmeier and Peter Daniel. Vol. 7612. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 210–222. ISBN: 978-3-642-33677-5. DOI: 10.1007/978-3-642-33678-2_18. (Cit. on p. 27).

[The16] The Coq Development Team. *The Coq Proof Assistant Reference Manual*. Version 8.5. INRIA. Jan. 2016. URL: https://coq.inria.fr/distrib/8.5/files/Reference-Manual.pdf (cit. on pp. 10, 127).

[Wen16] Makarius Wenzel. *The Isabelle/Isar Reference Manual*. Feb. 2016. URL: https://isabelle.in.tum.de/doc/isar-ref.pdf (cit. on p. 127).

[Wey+12] Danny Weyns, M. Usman Iftikhar, Didac Gil de la Iglesia, and Tanvir Ahmad. "A Survey of Formal Methods in Self-adaptive Systems". In: *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. C3S2E '12. Montreal, Quebec, Canada: ACM, 2012, pp. 67–79. ISBN: 978-1-4503-1084-0. DOI: 10.1145/2347583.2347592. (Cit. on p. 27).

[WLF01] Michel Wermelinger, Antónia Lopes, and José Luiz Fiadeiro. "A Graph Based Architectural (Re)Configuration Language". In: *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ESEC/FSE-9. Vienna, Austria: ACM, 2001, pp. 21–32. ISBN: 1-58113-390-1. DOI: 10.1145/503209.503213. (Cit. on p. 27).

[Wor08] Workflow Management Coalition. *Process Definition Interface – XML Process Definition Language*. Oct. 2008. URL: http://www.xpdl.org/standards/xpdl-2.1/WFMC-TC-1025-Oct-03-08-2-1.pdf (cit. on p. 29).

A formal approach for correct-by-construction system substitution

Safety-critical systems depend on the fact that their software components provide services that behave correctly (*i.e.* satisfy their requirements). Additionally, in many cases, these systems have to be adapted or reconfigured in case of failures or when changes in requirements or in quality of service occur. When these changes appear at the software level, they can be handled by the notion of substitution. Indeed, the software component of the source system can be substituted by another software component to build a new target system. In the case of safety-critical systems, it is mandatory that this operation enforces that the new target system behaves correctly by preserving the safety properties of the source system during and after the substitution operation.

In this thesis, the studied systems are modeled as state-transition systems. In order to model system substitution, the Event-B method has been selected as it is well suited to model such state-transition systems and it provides the benefits of refinement, proof and the availability of a strong tooling with the Rodin Platform.

This thesis provides a generic model for system substitution that entails different situations like cold start and warm start as well as the possibility of system degradation, upgrade or equivalence substitutions. This proposal is first used to formalize substitution in the case of discrete systems applied to web services compensation and allowed modeling correct compensation. Then, it is also used for systems characterized by continuous behaviors like hybrid systems. To model continuous behaviors with Event-B, the Theory plug-in for Rodin is investigated and proved successful for modeling hybrid systems. Afterwards, a correct substitution mechanism for systems with continuous behaviors is proposed. A safety envelope for the output of the system is taken as the safety requirement. Finally, the proposed approach is generalized, enabling the derivation of the previously defined models for web services compensation through refinement, and the reuse of proofs across system models.

Keywords: formal methods, correct-by-construction systems, system substitution, refinement

Une approche formelle pour la substitution correcte par construction de systèmes

Les systèmes critiques dépendent du fait que leurs composants logiciels fournissent des services aux comportements corrects (c'est-à-dire satisfaisant leurs exigences). De plus, dans de nombreux cas, ces systèmes doivent être adaptés ou reconfigurés en cas de pannes ou quand des évolutions d'exigences ou de qualité de service se produisent. Quand ces évolutions peuvent être capturées au niveau logiciel, il devient possible de les traiter en utilisant la notion de substitution. En effet, le composant logiciel du système source peut être substitué par un autre composant logiciel pour construire un nouveau système cible. Dans le cas de systèmes critiques, cette opération impose que le nouveau système cible se comporte correctement en préservant, autant que possible, les propriétés de sécurité et de sûreté du système source pendant et après l'opération de substitution.

Dans cette thèse, les systèmes étudiés sont modélisés par des systèmes états-transitions. Pour modéliser la substitution de systèmes, la méthode Event-B a été choisie car elle est adaptée à la modélisation de systèmes états-transitions et permet de bénéficier des avantages du raffinement, de la preuve et de la disponibilité d'un outil puissant avec la plate-forme Rodin.

Cette thèse fournit un modèle générique pour la substitution de systèmes qui inclut différentes situations comme le démarrage à froid et le démarrage à chaud, mais aussi la possibilité de dégradation ou d'extension de systèmes ou de substitution équivalente. Cette approche est d'abord utilisée pour formaliser la substitution dans le cas de systèmes discrets appliqués à la compensation de Services Web. Elle permet de modéliser la compensation correcte. Par la suite, cette approche est mise en œuvre dans le cas des systèmes caractérisés par des comportements continus comme les systèmes hybrides. Pour modéliser des comportements continus avec Event-B, l'extension Theory pour Rodin est examinée et s'avère performante pour modéliser des systèmes hybrides. Cela nous permet de proposer un mécanisme de substitution correct pour des systèmes avec des comportements continus. L'exigence de sûreté devient alors le maintien de la sortie du système dans une enveloppe de sûreté. Pour finir, l'approche proposée est généralisée, permettant la dérivation des modèles précédemment définis pour la compensation de Services Web par le raffinement et la réutilisation de preuves entre des modèles de systèmes.

Mots-clés : méthodes formelles, systèmes corrects par construction, substitution de systèmes, raffinement