 Electronic Notes in Theoretical Computer Science 215 (2008) 93–110 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Residual for Component Specifications

# Jean-Baptiste Raclet[1](#_bookmark0)

*IRISA*

*Campus de Beaulieu*

*F-35042 Rennes Cedex, France*

Abstract

We address the problem of component reuse by describing a quotient operation. Starting from the specifications of the behaviors of the component and of the desired overall system, this operation computes the residual specification characteristic of the systems that, when composed with the given component, satisfy the overall specification. This problem is solved when behaviors are given by modal or acceptance specifications and when composition allows mixed product and internalization of events. We show on an example how weak form of liveness constraint may be taken into account by this technique.

*Keywords:* Component-based design, Behavioral interface, Modal and acceptance specifications, Residual of specifications, Behavioral reuse.

# Introduction

In current component platforms, a component is equipped with an interface which lists the signature of the services that the entity offers. This light de- scription is sufficient to enable component reuse. However, it provides no guarantee that the reused component will interact suitably with its environ- ment and critical behavioral mismatch such as deadlock may occur.

In this paper, we investigate the extension of component interfaces to be- havioral descriptions in order to provide techniques to reason about component reuse at a behavioral level rather than at a signature level. More precisely, we study the following issues: can a component, the behavior of which is described in its interface by the specification S1, be used to build a system satisfying a

1 Email: [Jean-Baptiste.Raclet@irisa.fr](mailto:Jean-Baptiste.Raclet@irisa.fr)

1571-0661© 2008 Elsevier B.V. Open access under [CC BY-NC-ND license](http://creativecommons.org/licenses/by-nc-nd/3.0/).

doi:10.1016/j.entcs.2008.06.023

global specification S ? If so, what are the components that, when composed with the reused component, constitute a composite system satisfying S?

These problems can be seen as kinds of supervisor synthesis with the main difference that the reused component (corresponding to the plant in control theory) is a black-box. Indeed, a component must be reusable from the de- scription of its behavior in its interface and not from its implementation which is unknown as it may have been developed by a third party.

Behavioral reuse of components can also be related to some works in equa- tion solving. This problem introduced in [[11](#_bookmark33)] consists in solving the equation S1 × *X* S with S1 called the context, S a global specification and a trace equivalence relation. Solutions for this problem were proposed for various models of specification: finite automata [[11](#_bookmark33),[5](#_bookmark27)], finite state machine [[14](#_bookmark36)] (with inclusion of traces as equivalence relation), CCS or CSP processes [[12](#_bookmark34),[10](#_bookmark32)] (with bisimulation as equivalence relation) or input/output automata [[8](#_bookmark30)].

In this paper, we introduce modal specifications and acceptance specifi- cations as intuitive formalisms for behavioral interface description. From the expressiveness point of view, they allow to state some forms of liveness prop- erties. For each of this formalism, a quotient operation is defined to address the problem of behavioral reuse of a component as computing the residual specification S*/*S1. We study two kinds of composition between components: synchronous composition and mixed product with internalization of events. This paper is organized as follows: after short preliminaries, section [3](#_bookmark1) intro- duces modal specification. The notion of residual specification for behavioral reuse is formalized in section [4](#_bookmark7). Then, the quotient of modal specification is proposed in section [5](#_bookmark8). Expressivity is improved thanks to acceptance spec- ifications in section [6](#_bookmark14). Then, section [7](#_bookmark19) is devoted to the quotient operation corresponding to the use of mixed product with internalization of events as composition operation. An example and a hint for some line of future work conclude the paper.

# Preliminaries

We use Σ to denote the universe of events. Σ∗ denotes the set of all finite length event sequences, called traces, with *ϵ* the zero length trace. A language *L* over Σ is a subset of Σ∗. For *u* ∈ *L*, we let *Lu* be the set of events *a* such that the trace *u* followed by *a* (noted *u.a*) belong to *L*. The operations ∩ and

∪ over languages correspond to the set-theoretic intersection and union. The complement of the language *L* over Σ, noted ¬*L*, is the set Σ∗ \ *L*. Given a trace *w* ∈ Σ' and a subset of events Σ ⊆ Σ', the projection of *w* on Σ denoted ΠΣ(*w*) is the trace obtained from *w* by erasing the events not belonging to Σ.

A trace *u* is said to be a prefix of a trace *w* (noted *u* ≤ *w*), if *w* = *uv*. Given a language *L* its prefix closure *L*˜ consists of all the prefixes of all the traces in

*L*. A language *L* is said prefix-closed when *L* = *L*˜.

# Modal specifications

In this section we introduce modal specification as a formalism to specify sets of languages:

* 1. *Modal speciﬁcation and their models*

Modal automata are standard finite automata with modalities ”may” or ”must” on transitions. They were originally used in [[9](#_bookmark31)] to study the refinement of actions. We now introduce modal specifications which generalize the use of modalities on events to non-necessarily regular languages:

Definition 3.1 A modal specification S over Σ is a triple ⟨*L, must, may*⟩ where *L* is a prefix-closed language over Σ and *must, may* : *L* → P(Σ) are partial functions that type events: for *u* ∈ *L*,

* + - *a* ∈ *may*(*u*) means that the event *a* is allowed after the trace *u*;
    - *a* ∈ *must*(*u*) means that the event *a* is required after the trace *u*;
    - *a* ∈*/ may*(*u*) (often denoted *a* ∈ *mustnot*(*u*)) means that *a* is forbidden after

*u*.

For consistency, the following conditions are required for all *u* ∈ *L*: (C1) *must*(*u*) ⊆ *may*(*u*);

(C2) *may*(*u*)= *Lu*.

The consistency condition (C1) expresses that the events required after a trace *u* of the specification must also be allowed.

In the sequel the elements of the tuple corresponding to the modal spec- ification S could be denoted *L*(S), *must*(S) and *may*(S). The prefix-closed language *L*(S) may be called the *support* of S. The set of modal specifications over Σ is denoted MS(Σ).

A model of a modal specification is a language. The definition of the validation relation is the following:

Definition 3.2 A prefix-closed language *C* ⊆ Σ∗ is a model of the modal specification S ∈ MS(Σ), noted *C* |= S, if:

* + - *C* ⊆ *L*(S);
    - for all *u* ∈ *C*, *must*(S)(*u*) ⊆ *Cu* ⊆ *may*(S)(*u*).

The interpretation of this definition is the following: every trace *u* of a model *C* is also a trace of S and the events available in *C* after a trace *u* are all the required events (*must*(*u*) ⊆ *Cu*) and none of the forbidden events (*Cu* ⊆ *may*(*u*)) after the trace *u* in the specification.

Example 3.3 The specification T = ⟨Σ∗*, must, may*⟩ with, for all *u* ∈ Σ∗,

*must*(*u*) = ∅ and *may*(*u*) = Σ, admit every language over Σ as model. ⊥ =

⟨∅*,* ∅*,* ∅⟩ has no model.

Modal specifications are ordered by the following relation:

Definition 3.4 S1 ≤ S2 if and only if:

* *L*(S1) ⊆ *L*(S2) and

⎧⎨

*may*(S1)(*u*) ⊆ *may*(S2)(*u*)

* ∀*u* ∈ *L*(S )*,*

1

⎩ *must*(S1)(*u*) ⊇ *must*(S2)(*u*)

Remark 3.5 Any prefix-closed language *C* can be viewed as a modal speci- fication where *may*(*u*)= *must*(*u*)= *Cu* for all trace *u* ∈ *C*. Hence *C* |= S if and only if *C* ≤ S.

Proposition 3.6 S1 ≤ S2 *if and only if every model C of* S1 *is also a model of* S2*.*

When *L*(S) is regular, S can be viewed as the unfolding of a modal au- tomaton [[9](#_bookmark31)] all of whose states are final. The logical fragment equivalent to modal automata has been identified in [[6](#_bookmark28)]. It is a fragment of the mu-calculus called the conjunctive nu-calculus as it includes conjunctions and greatest fix- points along with diamond and box modalities. It is strictly less expressive than mu-calculus as neither disjunction nor eventualities can be stated.

* 1. *Pseudo-modal speciﬁcations*

For technical reasons, we shall consider an extension of the class of modal specification called *pseudo-modal speciﬁcation* and denoted *p*MS. They are specifications where the consistency condition (C1) of the definition [3.1](#_bookmark2) is relaxed for some traces *u*:

Definition 3.7 A pseudo-modal specification *p*S ∈ *p*MS(Σ) is a triple

⟨*L, must, may*⟩ where *L* is a prefix-closed language over Σ and *must, may* : *L* → P(Σ) are partial functions that type events with *no* consistency con- straint between *may*(*u*) and *must*(*u*).

A trace *u* of *p*S such that *must*(*u*) ¢ *may*(*u*) is said incoherently specified.

The relations |= and ≤ for pseudo-modal specifications are the same as for modal specifications (cf. def. [3.2](#_bookmark3) and [3.4](#_bookmark4)). Hence, if *must*(*u*) ¢ *may*(*u*) then *u* can’t belong to a language *C* model of *p*S because this would imply, for *a* ∈ *must*(*u*) and *a* ∈*/ may*(*u*), on one hand that *u.a* ∈ *C* (as *a* ∈ *must*(*u*)) and, on the other hand, that *u.a* ∈*/ C* (as *a* ∈*/ may*(*u*)). This remark give the intuition of a reduction of a pseudo-modal specification into a modal specification with respect to its set of models; we let *ρ* : *p*MS → MS be the operation such that:

Proposition 3.8 *Either a pseudo-modal speciﬁcation p*S *has no model or there exists a largest modal speciﬁcation ρ*(*p*S) *smaller than p*S*, and ρ*(*p*S) *has the same models as p*S*.*

The modal specification *ρ*(*p*S) is obtained from the pseudo-modal specifi- cation *p*S by application of the following steps:

1. basis: we let *R* be a copy of *p*S;
2. we let *U* be the set of traces *u* incoherently specified in *R*; we remove *U* from *L*(*R*); for all trace *v* such that *v.a* = *u* with *u* ∈ *U* , we remove *a* from *may*(*R*)(*v*) to enforce the consistency condition (*C*2);

When *a* ∈ *must*(*R*)(*v*), *v* becomes incoherently specified in *R*. Thus, we repeat this step until there is no more incoherently specified trace in *L*(*R*);

1. *ρ*(*p*S) is built from *R*: *L*(*ρ*(*p*S)) =

*L*˜(*R*) and for all *u* ∈ *L*(*ρ*(*p*S)),

*may*(*ρ*(*p*S))(*u*)= *may*(*R*)(*u*) and *must*(*ρ*(*p*S))(*u*)= *must*(*R*)(*u*).

* 1. *Lattice of modal speciﬁcations*

The set of modal specifications MS equipped with the partial order ≤ is a complete distributive lattice (hence a bounded lattice) where the meet S1 ∧ S2 is the reduction of the pseudo-modal specification S1&S2 over *L*(S1) ∩ *L*(S2) with:

⎧⎨

*may*(S1&S2)(*u*) = *may*(S1)(*u*) ∩ *may*(S2)(*u*)

∀*u* ∈ *L*(S ) ∩ *L*(S )*,*

1 2

⎩

*must*(S1&S2)(*u*) = *must*(S1)(*u*) ∪ *must*(S2)(*u*)

and the join S1 ∨ S2 is the modal specification over *L*(S1) ∪ *L*(S2) with:

⎪ 1 2 1 2

⎧⎪ if *u* ∈ *L*(S1) ∩ *L*(S2)*, may*(S1 ∨ S2)(*u*)= *may*(S1)(*u*) ∪ *may*(S2)(*u*)*,*

*must*(S

⎪

∨S )(*u*)= *must*(S )(*u*) ∩ *must*(S )(*u*)

⎪⎨ if *u* ∈ *L*(S1) \ *L*(S2)*, may*(S1 ∨ S2)(*u*)= *may*(S1)*, must*(S1 ∨ S2)(*u*)

= *must*(S1)(*u*)

⎪

⎪

if *u* ∈ *L*(S2) \ *L*(S1)*, may*(S1 ∨ S2)(*u*)= *may*(S2)*, must*(S1 ∨ S2)(*u*)

⎪⎩ = *must*(S2)(*u*)*.*

Proposition 3.9 *For all modal speciﬁcation* S ∈ MS(Σ)*, we have:*

S = {C | C |= S}

# Residual specification for behavioral reuse

In the sequel, a component is a pair (C*,* S) such that C |= S with C called the implementation and S the specification of the component. Reusing a component (C1*,* S1) to realize a global system specified by S amounts to exhibit a residual specification S*/*S1 so that any component (C2*,* S*/*S1) is such that the composition of C1 with C2 (noted C1 ⊗ C2) satisfies S. In component-based design, components are regarded as black-box. As a result, the implementation C1 of the component to be reused is unknown and its composition with the possible components (C2*,* S*/*S1) must realize S whatever the implementation C1 of S1 could be. Thus the characteristic property of a residual operation for behavioral reuse of a component is the following:

Proposition 4.1 C2 |= S*/*S1 ⇔ ∀C1*.*[C1 |= S1 ⇒ C1 ⊗ C2 |= S]*.*

In the next section, we establish this proposition when S and S1 are modal specifications and when C1 and C2 are composed using a synchronous product.

# Quotient of modal specifications for behavioral reuse

The synchronous product of languages C1 and C2 over the same alphabet Σ corresponds to the set theoretic intersection C1 ∩ C2. We generalize the syn- chronous product to modal specifications:

Definition 5.1 The synchronous product of S1 and S2 is the modal specifi- cation S1 ⊗ S2 over *L*(S1) ∩ *L*(S2) with:

∀*u* ∈ *L*(S ) ∩ *L*(S )*,* ⎧⎨ *must*(S1 ⊗ S2)(*u*) = *must*(S1)(*u*) ∩ *must*(S2)(*u*)

1 2

⎩

*may*(S1 ⊗ S2)(*u*) = *may*(S1)(*u*) ∩ *may*(S2)(*u*)

Remark 5.2 (i) This operator is monotonic over the order relation ≤: S1 ≤ S2 ⇒ (S ⊗ S1 ≤S ⊗ S2 and S1 ⊗S ≤ S2 ⊗ S)

(ii) If C1*,* C2 ⊆ Σ∗ are viewed as modal specifications (cf. remark [3.5](#_bookmark5)) then

C1 ⊗ C2 = C1 ∩ C2.

Now, to define the quotient operation, we start from the observation that, for any prefix-closed languages *L*, *M*, and *N*: *L*∩*M* ⊆ *N* ⇔ *L* ⊆⇓ (*N* ∪ ¬*M*) where:

⇓ *X* = {*u* ∈ Σ∗ | ∀*v v* ≤ *u* ⇒ *v* ∈ *X* }

denote the prefix interior of a set *X*; it is an interior operation giving the greatest prefix-closed subset of the given set, when such a subset exists and the empty set otherwise. This remark is used to define the support of the modal specification S*/*S1. Now in order to define the typing functions *may*(S*/*S1) and *must*(S*/*S1), we proceed by case inspection:

Definition 5.3 The quotient of the modal specifications S and S1 is the pseudo-modal specification S*/*S1 over ⇓ (*L*(S) ∪ ¬*L*(S1)) with:

1. for all *u* ∈ *L*(S*/*S1) ∩ *L*(S1):
   1. if *a* ∈ *must*(S)(*u*) ∩ *must*(S1)(*u*) then *a* ∈ *must*(S*/*S1)(*u*) and *a* ∈

*may*(S*/*S1)(*u*);

* 1. if *a* ∈ *must*(S)(*u*) ∩ ¬*must*(S1)(*u*) then *a* ∈ *must*(S*/*S1)(*u*) but *a* ∈*/ may*(S*/*S1)(*u*);
  2. if *a* ∈ (*may*(S)(*u*) \ *must*(S)(*u*)) then *a* ∈ *may*(S*/*S1)(*u*);
  3. if *a* ∈ *mustnot*(S)(*u*) ∩ *mustnot*(S1)(*u*) then *a* ∈ *may*(S*/*S1)(*u*);
  4. if *a* ∈ *mustnot*(S)(*u*) ∩ *may*(S1)(*u*) then *a* ∈ *mustnot*(S*/*S1)(*u*);

1. if *u* ∈ (*L*(S*/*S1) \ *L*(S1)) then *must*(S*/*S1)(*u*)= ∅ and *may*(S*/*S1)(*u*)= Σ.

Now we give for each possible case, an intuitive interpretation of the re- sulting modality assuming that C1 |= S1 and we intend to have C2 |= S*/*S1 and C1 ⊗ C2 |= S:

1. first, when *u* ∈ *L*(S*/*S1) ∩ *L*(S1):
   1. *a* is required in the global specification S that is *u.a* must belong to C1 ⊗ C2 = C1 ∩ C2. As *a* is guaranteed in S1, *u.a* ∈ C1 for all C1 |= S1 with *u* ∈ C1; thus *u.a* must belong to C2 to always have *u.a* ∈ C1 ⊗ C2: *a* ∈ *must*(S*/*S1)(*u*) and *a* ∈ *may*(S*/*S1)(*u*).
   2. *a* is required in the global specification S but as *a* ∈*/ must*(S1)(*u*),

there are some C1 |= S1 such that *u.a* ∈*/* C1; hence, for all C2, C1 ⊗ C2

can’t be a model of S. As a result, the trace *u* must be incoherently

in S*/*S1 and we let *a* ∈ *must*(S*/*S1)(*u*) but *a* ∈*/*

model this inconsistency.

*may*(S*/*S1)(*u*) to

* 1. *a* is allowed in the global specification S and *u.a* may belong to C1⊗C2. Thus, whether or not *u.a* belongs to C1 |= S1, *u.a* can belong to C2 without violating the specification S. Hence: *a* ∈ *may*(S*/*S1)(*u*).
  2. *a* is forbidden in the global specification S and in S1 thus, whether or not *u.a* ∈ C2, we have *u.a* ∈*/* C1 ⊗ C2 which is conform to S. Hence: *a* ∈ *may*(S*/*S1)(*u*).
  3. *a* is forbidden in the global specification S. As there are some C1 |= S1 with *u.a* ∈ C1, we forbid *a* in S*/*S1: *a* ∈ *mustnot*(S*/*S1)(*u*). As a result, when C2 |= S*/*S1, *u.a* ∈*/* C1 ⊗ C2 which is conform to S.

1. if *u* ∈ (*L*(S*/*S1) \ *L*(S1)): as *u* ∈*/* C1, *u* may belong to C2, it won’t belong to C1 ⊗ C2. As a result, S*/*S1 is relaxed after the trace *u* by taking *must*(S*/*S1)(*u*)= ∅ (nothing is required) and *may*(S*/*S1)(*u*) = Σ (every event is allowed).

The adjoint operation of this quotient operation is the synchronous product of definition [5.1](#_bookmark10):

Proposition 5.4 *If* S*,* S1 *and* S2 *are modal speciﬁcations over* Σ *then:*

S1 ⊗ S2 ≤ S ⇔ S2 ≤ *ρ*(S*/*S1)

To prove proposition [4.1](#_bookmark9) for modal specifications and synchronous product, we need the following lemma:

Lemma 5.5 ∨ {C ⊗ C' |C |= S} = ∨ {C | C |= S} ⊗ C'

Proposition 5.6 *If* S *and* S1 *are modal speciﬁcations over* Σ *then:*

C2 |= S*/*S1 *iff* ∀C1*.*[C1 |= S1 ⇒ C1 ⊗ C2 |= S]

Proof. (⇒) According to Prop. [5.4](#_bookmark12), if C2 |= S*/*S1 (that is C2 |= *ρ*(S*/*S1)) then C2 ⊗ S1 ≤ S. Moreover as C1 |= S1 then C1 ⊗ C2 ≤ S1 ⊗ C2. As a result, C1 ⊗ C2 ≤S that is C1 ⊗ C2 |= S.

(⇐) If for all C1 such that C1 |= S1 we have C1 ⊗ C2 |= S then:

∨ {C1 ⊗ C2 | C1 |= S1 } ≤ S

Thus, by lemma [5.5](#_bookmark13), ∨ {C1 | C1 |= S1 } ⊗ C2 ≤ S i.e. S1 ⊗ C2 ≤ S (by Prop. [3.9](#_bookmark6)). According to Prop. [5.4](#_bookmark12), C2 ≤ S*/*S1 hence C2 |= S*/*S1.

As previously pointed out, the disjunction is not included in the logical fragment equivalent to modal automata. Therefore particular liveness prop- erties can’t be stated in this framework. For instance, let us consider the following ”progressive” property: ”any stimulus *a* is followed by *at least b*1 or *b*2 as reaction”. This can’t be specified with a modal specification: *b*1 and *b*2 can’t belong to *must*(”*a*”) because this would request that every stimulus *a* is followed by both *b*1 and *b*2; *b*1 and *b*2 can’t also belong to *may*(”*a*”) because the language such that the stimulus *a* is followed by no reaction would be a model of the specification.

A trace *u* in a modal specification specifies any situation where the system is ready to engage in a set of events *X*, if and only when *must*(*u*) ⊆ *X* ⊆ *may*(*u*). This set of ”acceptance” set is thus given by:

*Acc*(*u*)= {*X* ∈ P(Σ) | *must*(*u*) ⊆ *X* ⊆ *may*(*u*) }

By definition this set is closed under union, intersection and convexity (that is given *X, Y* ∈ *Acc*(*u*) and a set *Z* such that *X* ⊆ *Z* ⊆ *Y* then *X* ∪ *Y* , *X* ∩ *Y* and *Z* ∈ *Acc*(*u*)) and may and must modalities may be recovered as *may*(*u*)= *X*∈*Acc*(*u*) *X* and *must*(*u*)= *X*∈*Acc*(*u*) *X*.

Thus, for example if *may*(*u*) = {*b*1*, b*2} and *must*(*u*) = ∅, we obtain *Acc*(*u*) = {∅*,* {*b*1}*,* {*b*2}*,* {*b*1*, b*2}}. If we want to specify that at least *b*1 or *b*2 occur, the specified acceptance set should be {{*b*1}*,* {*b*2}*,* {*b*1*, b*2}} which is no longer closed by intersection. According to this example, closure by inter- section should be relaxed to deal with such ”progressive” properties. Trees

labeled by acceptance set closed by union and convexity have been studied in [[7](#_bookmark29)]. In the next section, we propose a quotient operation for acceptance specifications with no closure constraint over the acceptance set.

# Improving expressivity with acceptance spec

We generalize the previous framework presented for modal specifications to acceptance specifications:

* 1. *Acceptance speciﬁcations and their models*

Definition 6.1 An acceptance specification S is a pair S = ⟨*L, Acc*⟩ where *L* is a prefix-closed language over Σ and *Acc* : *L* → P(P(Σ)) is a map associating each trace *u* ∈ *L* to its acceptance set. For consistency, we require for all trace *u* ∈ *L*:

(C1) *Acc*(*u*) /= ∅

(C2) *u.a* ∈ *L* if and only there exists at least one set *X* ∈ *Acc*(*u*) such that

*a* ∈ *X*

The condition (C2) can be rephrase in *Lu* = *X*∈*Acc*(*u*) *X*. The set of acceptance specifications over Σ is denoted AS(Σ).

Definition 6.2 A prefix-closed language *C* ⊆ Σ∗ is a model of the acceptance specification S ∈ AS(Σ), noted *C* |= S, if:

* *C* ⊆ *L*(S);
* for all *u* ∈ *C*, *Cu* ∈ *Acc*(S)(*u*).

Example 6.3 The specification T = ⟨Σ∗*, Acc*⟩ with, for all *u* ∈ Σ∗, *Acc*(*u*)=

P(Σ), admit every language over Σ as model. ⊥ = ⟨∅*,* ∅⟩ has no model.

Remark 6.4 *Acc*(*u*) = ∅ is different from ∅ ∈ *Acc*(*u*). The first situation is a violation of a consistency condition whereas the second reports that some models of the specification can perform no event after the trace *u*.

Definition 6.5 The order relation on acceptance specifications is given by inclusion of both corresponding languages and acceptance sets:

S1 ≤ S2 iff *L*(S1) ⊆ *L*(S2) and ∀*u* ∈ *L*(S1)*, Acc*(S1)(*u*) ⊆ *Acc*(S2)(*u*)

Remark 6.6 Any language *C* can be viewed as an acceptance specification with *Acc*(*u*)= *Cu* that is its acceptance set is a singleton for all trace *u* ∈ *C*. Hence *C* |= S if and only if *C* ≤ S.

* 1. *Pseudo-acceptance speciﬁcations*

Definition 6.7 A pseudo-acceptance specification *p*S ∈ *p*AS(Σ) is a pair

⟨*L, Acc*⟩ where *L* is a language over Σ and *Acc* : *L* → P(P(Σ)) is a map associating each trace *u* to its set of acceptance with *no* consistency constraint over *Acc*.

A trace *u* of *p*S is said incoherently specified when *Acc*(*u*)= ∅.

Every pseudo-acceptance specification *p*S can be reduced to an acceptance specification *ρ*(*p*S) that admits the same models, by iteration of the following steps:

1. basis: we let *R* be a copy of *p*S;
2. we let *U* be the set of traces *u* incoherently specified in *R*; we remove *U* from *L*(*R*); for all trace *v* such that *v.a* = *u* with *u* ∈ *U* , we remove from *Acc*(*R*)(*v*) all sets containing the letter *a* to enforce the consistency condition (*C*2);

When *a* ∈ *X*∈*Acc*(*R*)(*v*) *X*, *v* becomes incoherently specified in *R*. Thus, we repeat this step until there is no more incoherently specified trace in *L*(*R*);

1. *ρ*(*p*S) is built from *R*: *L*(*ρ*(*p*S)) =

*Acc*(*ρ*(*p*S))(*u*)= *Acc*(*R*)(*u*).

*L*˜(*R*) and for all *u* ∈ *L*(*ρ*(*p*S)),

* 1. *Lattice of acceptance speciﬁcations*

The set of acceptance specifications AS equipped with the partial order ≤ is a complete distributive lattice (hence a bounded lattice) where the meet S1 ∧ S2 is the reduction of the pseudo-acceptance specification S1&S2 whose support is *L*(S1) ∩ *L*(S2) and with *Acc*(S1&S2)(*u*) = *Acc*(S1)(*u*) ∩ *Acc*(S2)(*u*). The join S1 ∨ S2 is defined over *L*(S1) ∪ *L*(S2) by:

⎧⎪ if *u* ∈ *L*(S1) ∩ *L*(S2)*, Acc*(S1 ∨ S2)(*u*)= *Acc*(S1) ∪ *Acc*(S2)

⎪⎨ if *u* ∈ *L*(S ) \ *L*(S )*, Acc*(S ∨ S )(*u*)= *Acc*(S )

1 2 1 2 1

⎪⎪⎩ if *u* ∈ *L*(S2) \ *L*(S1)*, Acc*(S1 ∨ S2)(*u*)= *Acc*(S2)

* 1. *Quotient of acceptance speciﬁcations*

We define quotient of acceptance specifications such that proposition [4.1](#_bookmark9) is verified with synchronous product as component composition:

Definition 6.8 The synchronous product of the acceptance specifications S1 and S2 is the acceptance specification S1 ⊗ S2 over *L*(S1) ∩ *L*(S2) with for all *u* ∈ *L*(S1) ∩ *L*(S2):

*Acc*(S1 ⊗ S2)(*u*)= {*X*1 ∩ *X*2 | *X*1 ∈ *Acc*(S1)(*u*) and *X*2 ∈ *Acc*(S2)(*u*)}

This operation has for adjoint the following quotient operation:

Definition 6.9 The quotient of the acceptance specification S and S1 is the pseudo-acceptance specification S*/*S1 over ⇓ (*L*(S) ∪ ¬*L*(S1)) with, for all *u* ∈ *L*(S*/*S1) ∩ *L*(S1): *Acc*(S*/*S1)(*u*) =

{*Y* ∈ P(Σ) | ∀*X* ∈ *Acc*(S1)(*u*)*,X* ∩ *Y* ∈ *Acc*(S)(*u*) }

and for all *u* ∈ (*L*(S*/*S1) \ *L*(S1)), *Acc*(S*/*S1)(*u*)= P(Σ).

Proposition 6.10 *If* S*,* S1 *and* S2 *are acceptance speciﬁcations over* Σ *then:*

S1 ⊗ S2 ≤ S ⇔ S2 ≤ *ρ*(S*/*S1)

Similarly to the proof for modal specifications, we use the previous result to establish the characteristic property of a residual operation for behavioral reuse of a component:

Proposition 6.11 *If* S *and* S1 *are acceptance speciﬁcations then:*

C2 |= S*/*S1 *iff* ∀C1*.*[C1 |= S1 ⇒ C1 ⊗ C2 |= S]

As previously briefly noticed, acceptance specifications strictly subsumes modal specifications. Indeed, consider the two following transformations:

Definition 6.12 Let S = (*L, must, may*) ∈ MS and S' = (*L*'*, Acc*') ∈ AS:

* *j* : MS → AS

*j*(S)= (*L, Acc*) with *Acc*(*u*)= {*X* ∈ P(Σ) | *must*(*u*) ⊆ *X* ⊆ *may*(*u*) }

* *k* : AS → MS

*k*(S')= (*L*'*, must*'*, may*') with

⎧⎨ *may*'(*u*') = *X*∈*Acc*'(*u*') *X*

⎩ *must*'(*u*') = *X*∈*Acc*'(*u*') *X*

We have: *k* ◦ *j* = *Id* but *j* ◦ *k* /= *Id*. Quotient operations for modal specifications and acceptance specifications can be related:

Proposition 6.13 *The quotient operation for modal speciﬁcations is a par- ticularization of the quotient operation for acceptance speciﬁcations.*

Proof. Given S and S1 two modal specifications, we let S' be the accep- tance specifications obtained by quotienting *j*(S) and *j*(S1) using definition

[6.9](#_bookmark16). Then the modal specification *k*(S') is identical to the one obtained by quotienting S and S1 using definition [5.3](#_bookmark11).

Synchronous product is a very restrictive form of composition. In proposi- tion [6.11](#_bookmark18), the component (C2*,* S*/*S1) restrict the behavior of all possible (C1*,* S1) so that the composite system C1 ⊗ C2 realizes S. In the next section, we in- vestigate an approach where the component (C2*,* S*/*S1) may also contribute directly to the realization of the specification S: given S a global specification over Σ and (C1*,* S1) a component to be reused over Σ1, the events belonging to Σ \ Σ1 are realized by the component (C2*,* S*/*S1). Thus, we now consider residual of specifications when component composition corresponds to mixed product [[4](#_bookmark26)]. We also consider internalization of event that is C1 and C2 may evolved without being observed externally.

# Using mixed product with internalization of events as component composition

We first recall the definition of mixed product, restriction and expansion of languages:

Definition 7.1 • Given C1 and C2 two languages respectively over Σ1 and Σ2, the mixed product of C1 and C2 is the language:

C1HHC2 = {*w* ∈ (Σ1 ∪ Σ2)∗ | ΠΣ1 (*w*) ∈ C1 and ΠΣ2 (*w*) ∈ C2 }

* Given C a language over Σ' and Σ ⊆ Σ', the restriction of C to the alphabet Σ is the language:

C↓Σ = {*u* ∈ Σ∗ | *u* = ΠΣ(*w*) with *w* ∈ C }

* Given C a language over Σ and Σ ⊆ Σ', the expansion of C to the alphabet Σ' is the language:

C↑Σ' = {*w* ∈ Σ'∗ | ΠΣ(*w*) ∈ C }

Now, we generalize these operations for acceptance specifications:

* 1. *Mixed product of acceptance speciﬁcations*

Definition 7.2 The mixed product of the acceptance specifications S1 ∈ AS(Σ1) and S2 ∈ AS(Σ2) is the acceptance specifi- cation S1HHS2 over *L*(S1)HH*L*(S2) with for all *u* ∈ *L*(S1)HH*L*(S2):

*Acc*(S1HHS2)(*w*) = {(*X*1 ∪ (Σ2 \ Σ1)) ∩ (*X*2 ∪ (Σ1 \ Σ2)) |

*X*1 ∈ *Acc*(S1)(ΠΣ1 (*w*)) and *X*2 ∈ *Acc*(S2)(ΠΣ2 (*w*))}

When Σ1 = Σ2 the definition of the synchronous product is retrieved.

The mixed product of acceptance specifications can be related in a general way to the synchronous product:

Definition 7.3 Given Σ ⊆ Σ' and S ∈ AS(Σ), the *τ* -expansion of S to Σ' is the acceptance specification *τ*Σ' (S) over (*L*(S))↑Σ' with:

*Acc*(*τ*Σ' (S))(*w*)= {*Y* ∪ (Σ' \ Σ) | *Y* ∈ *Acc*(S)(ΠΣ(*w*))}

This operation consists in saturating the element of each acceptance set with all events of (Σ' \ Σ). Thus, mixed product of acceptance specifications is reduced to synchronous product (cf. definition [6.8](#_bookmark15)):

Proposition 7.4 *Given* S1 ∈ AS(Σ1) *and* S2 ∈ AS(Σ2)*:*

S1HHS2 = *τ*Σ1 ∪Σ2 (S1) ⊗ *τ*Σ1 ∪Σ2 (S2)

The restriction operation adjoint of the *τ* -expansion is the following:

Definition 7.5 Given Σ ⊆ Σ' and S' ∈ AS(Σ'), the M-restriction of S' to Σ is the acceptance specification MΣ(S') over (*L*(S))↓Σ with:

*Acc*(MΣ(S'))(*u*)= {*Y* | *Y* ∪ (Σ' \ Σ) ∈ {*Acc*(S')(*w*) | *w* ∈ *L*(S')

and ΠΣ(*w*)= *u*}}

Proposition 7.6 *Given* S ∈ AS(Σ) *and* S' ∈ AS(Σ') *with* Σ ⊆ Σ'*,*

*τ*Σ' (S) ≤ S' ⇔S ≤ MΣ(S')

To deal with internalization of event, we now define the restriction of an acceptance specification to a sub-alphabet:

* 1. *Restriction of acceptance speciﬁcation*

Definition 7.7 Given Σ ⊆ Σ' and S ∈ AS(Σ'), the restriction of S to Σ is the acceptance specification S↓Σ over (*L*(S))↓Σ with:

*Acc*(S↓Σ)(*u*)= {{*X* ∩ Σ | *X* ∈ *Acc*(S)(*w*) and

*X* ∩ Σ /= ∅ when *X* /= ∅} | *w* ∈ *L*(S) and ΠΣ(*w*)= *u*}

Definition 7.8 Given Σ ⊆ Σ' and S' ∈ AS(Σ), the expansion

of S' to Σ' is the acceptance specification S' ' over (*L*(S'))↑Σ' with:

↑Σ

*Acc*(S'

↑Σ

' )(*w*) = {*X* | *X* ∩ Σ ∈ *Acc*(S')(ΠΣ(*w*))}∪

{*X* | *X* /= ∅ and *X* ⊆ (Σ' \ Σ) }

This operation consists in allowing after each trace *u* of S' finite sequences of events in Σ' \ Σ.

Proposition 7.9 *Given* S ∈ AS(Σ') *and* S' ∈ AS(Σ) *with* Σ ⊆ Σ'*,*

S↓Σ ≤ S' ⇔S ≤ S' '

↑Σ

* 1. *Adjoint of the mixed product with internalization of events of acceptance speciﬁcation*

From the previous propositions, we can deduce:

(S1HHS2)↓Σ ≤S ⇔ S1HHS2 ≤ S↑(Σ1 ∪Σ2) by Prop. [7.9](#_bookmark22)

⇔ *τ*Σ1 ∪Σ2 (S1) ⊗ *τ*Σ1∪Σ2 (S2) ≤ S↑(Σ1 ∪Σ2 ) by Prop. [7.4](#_bookmark20)

⇔ *τ*Σ1 ∪Σ2 (S2) ≤ S↑(Σ1 ∪Σ2 )*/τ*Σ1 ∪Σ2 (S1) by Prop. [6.10](#_bookmark17)

⇔ S2 ≤ MΣ2 (S↑(Σ1 ∪Σ2 )*/τ*Σ1 ∪Σ2 (S1)) by Prop. [7.6](#_bookmark21) Similarly to the proof for modal specifications in the synchronous case,

we use the previous equivalence to establish the characteristic property of a residual operation for behavioral reuse of a component when the product of components is the mixed product with internalization of events:

Proposition 7.10 *Given* S ∈ AS(Σ)*,* S1 ∈ AS(Σ1) *and* Σ2 *such that* Σ ⊆

Σ1 ∪ Σ2*:*

∀C1*.*[C1 |= S1 ⇒ (C1HHC2)↓Σ |= S] ⇔ C2 |= MΣ2 (S↑(Σ1 ∪Σ2)*/τ*Σ1 ∪Σ2 (S1))

# An example

This example is inspired from [[3](#_bookmark24)]. In this paper, component interfaces are designed via interface automata. We refine the intended behavior thanks to *acceptance automata*; the quotient of acceptance specifications defined in [6.9](#_bookmark16) can be adapted when the support of the specification is a regular prefix-closed language [[13](#_bookmark35)].

* The goal is to build a system satisfying the following specification S over the alphabet Σ = {*msg, ok, f ail*}:

*msg*

|  |  |
| --- | --- |
|  | *Acc*(*S*) |
| 0’ | {{*msg*}} |
| 1’ | {{*ok*}} |



0’

1’

*ok*

* To realize S, we aim at reusing a component satisfying the following speci- fication S1 describing the behavior of a communication channel:



5

*ok*

*ack ack*

0

*msg*

1 *send*

2 *nack*

3

*send* 4

*f ail*

*nack*

6

|  |  |
| --- | --- |
|  | *Acc*(*S*1) |
| 0 | {{*msg*}} |
| 1 | {{*send*}} |
| 2 | {{*ack*}*,* {*nack*}*,* {*ack, nack*}} |
| 3 | {{*send*}} |
| 4 | {{*ack, nack*}} |
| 5 | {{*ok*}} |
| 6 | {{*f ail*}} |

S1 is defined over the alphabet Σ1 = {*msg, send, ack, nack,f ail, ok*}. Note that loss of message is allowed in *Acc*(S1)(2) which is not the case in *Acc*(S1)(4).

* We let Σ2 = {*ack, nack, send*}. As Σ \ Σ1 = ∅, (C2*,* S*/*S1) will restrict the behavior of (C1*,* S1) to enforce S. We have: *τ*Σ1 ∪Σ2 (S1)= S1 as Σ2 ⊆ Σ1.
* We compute *S*↑(Σ1 ∪Σ2 ):

*ack*



*msg*

*ack*

*nack send*

0’ 1’

*ok*

*nack send*

* *Acc*(*S*↑(Σ1 ∪Σ2))(0')= {{*msg*}*,* {*msg*}∪ *X, X*} with *X* ⊆ Σ2 and *X* /= ∅.
* *Acc*(*S*↑(Σ1 ∪Σ2))(1')= {{*ok*}*,* {*ok*}∪ *X, X*} with *X* ⊆ Σ2 and *X* /= ∅.
* Then we compute the quotient S↑(Σ1 ∪Σ2)*/τ*Σ1 ∪Σ2 (S1) (in the following figure, only transitions labeled by required events are depicted):



*ok*

51'

*ack ack*

*msg*

00'

11'

*send*

21'

*nack*

31'

*send*

41'

*nack*

61'

* *Acc*(00') = {{*msg*}∪ *X* | *X* ⊆ {*ok, f ail, ack, nack, send*} };
* *Acc*(11') = {{*send*}∪ *X* | *X* ⊆ {*ok, f ail, ack, nack, msg*}}
* *Acc*(21') = {{*ack, nack*}∪ *X* | *X* ⊆ {*ok, f ail, msg, send*}}
* *Acc*(31') = {{*send*}∪ *X* | *X* ⊆ {*msg, ok, f ail, ack, nack*}}
* *Acc*(41') = {{*ack*}∪ *X,* {*nack*}∪ *X,* {*ack, nack*}∪ *X*

| *X* ⊆ {*ok, f ail, msg, send*}}

* *Acc*(51') = {{*ok*}∪ *X* | *X* ⊆ {*msg,fail, ack, nack, send*}}
* *Acc*(61') = ∅
* Last, we apply the operation MΣ2 on the acceptance specification equivalent to the previous pseudo-acceptance specification. The result is the following:

a *send*

*send*

b *ack* d



*nack*

*ack, nack*

*send*

*ack*

c *send* e

*ack, nack*

*send ack, nack*

T

*ack, nack, send*

* *Acc*(*a*)= {{*send*}*,* {*send, ack*}*,* {*send, nack*}*,* {*send, ack, nack*}}
* *Acc*(*b*)= {{*ack, nack*}*,* {*ack, nack, send*}}
* *Acc*(*c*)= {{*send*}*,* {*send, ack*}*,* {*send, nack*}*,* {*send, ack, nack*}}
* *Acc*(*d*)= {{*send*}*,* {*send, ack*}*,* {*send, nack*}*,* {*send, ack, nack*}}
* *Acc*(*e*)= {{*ack*}*,* {*ack, send*}}
* *Acc*(T)= P(Σ2).

We remark that a sent message that has been acknowledged negatively *must* then be acknowledged positively (in state *e*, *ack* is required and *nack* is forbidden).

# Conclusion

In this paper, we have studied the problem of behavioral reuse of a compo- nent as the computation of a residual specification. We have introduced modal specification and acceptance specification as formalisms to specify component behavior. They allow to address restricted forms of liveness. Quotient of mu-calculus formulas was investigated in [[1](#_bookmark23)]. Mu-calculus is quite expressive but the complexity of the proposed quotient operation is double exponential in the size of the tree automata equivalent to the quotiented formulas. In contrast, our solutions using the automata-based version of modal and accep- tance specifications are polynomial [[13](#_bookmark35)]. Furthermore, to our knowledge, our approach is the first to consider components as black box in equation solving: the equation is solved for a given set of possible implementations characterized by a specification S1.

Future works concern the application of these techniques to the com- ponent adaptation problem. In particular, these techniques seems suited when detection of mismatch between components is performed thanks to the description of the properties the system should verify [[2](#_bookmark25)]. Moreover modal

and acceptance specifications are sets equipped with a lattice structure and a monoid structure with a residual operation, adjoint of a commutative product operation ie. are residuated lattices. We are interested in a more precise characterization of the underlying algebraic structure of the sets of modal and acceptance specification in order to develop the basis of an algebraic theory of components adaptation and reuse.

# Acknowledgement

The author would like to thank Eric Badouel and Philippe Darondeau for many suggestions and discussions.

# References

1. Arnold, A., A. Vincent and I. Walukiewicz, *Games for synthesis of controllers with partial observation*, Theoretical Computer Science 303 (2003), pp. 7–34.
2. Canal, C., J. M. Murillo and P. Poizat, *Software adaptation*, L’Objet 12 (2006), pp. 9–31.
3. de Alfaro, L. and T. A. Henzinger, *Interface automata*, in: *ESEC/FSE-9: Proceedings of the 8th* *European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering* (2001), pp. 109–120.
4. Duboc, C., *Mixed product and asynchronous automata.*, Theor. Comput. Sci. 48 (1986),

pp. 183–199.

1. El-Fakih, K., N. Yevtushenko, S. Buffalov and G. von Bochmann, *Progressive solutions to a parallel automata equation.*, Theoretical Computer Science 362 (2006), pp. 17–32.
2. Feuillade, G. and S. Pinchinat, *Modal speciﬁcations for the control theory of discrete event systems*, J. of Discrete Event Dynamic Systems 17 (2007).
3. Hennessy, M., *Acceptance trees*, J. ACM 32 (1985), pp. 896–928.
4. Kumar, R., S. Nelvagal and S. I. Marcus, *A discrete event systems approach for protocol* *conversion*, Discrete Event Dynamic Systems 7 (1997), pp. 295–315.
5. Larsen, K. G., *Modal speciﬁcations*, in: *Automatic Veriﬁcation Methods for Finite State* *Systems*, 1989, pp. 232–246.
6. Larsen, K. G. and L. Xinxin, *Equation solving using modal transition systems*, in: *Proceedings of the Fifth Annual IEEE Symp. on Logic in Computer Science, LICS 1990* (1990), pp. 108–117.
7. Merlin, P. and G. von Bochmann, *On the construction of submodule speciﬁcations and communication protocols*, ACM Trans. on Programming Languages and Systems 5 (1983),

pp. 1–25.

1. Parrow, J., *Submodule construction as equation solving in ccs*, Theoretical Computer Science

68 (1987), pp. 175–202.

1. Raclet, J.-B., *Residual for component speciﬁcations*, Research Report 6196, INRIA (2007).
2. Yevtushenko, N., T. Villa, R. K. Brayton, A. Petrenko and A. L. Sangiovanni-Vincentelli, *Solution of parallel language equations for logic synthesis*, in: *Proceedings of the International Conference on Computer-Aided Design*, 2001, pp. 103–110.