On Refinement of Generic State-based Software Components

Sun Meng[1] and Luís S. Barbosa[2]

[1] LMAM, School of Mathematical Science
Peking University, China
sunmeng@water.pku.edu.cn
[2] Department of Informatics
Minho University, Portugal
lsb@di.uminho.pt

Abstract. This paper characterizes refinement of state-based software components modelled as pointed coalgebras for some Set endofunctors. The proposed characterization is parametric on a specification of the underlying behaviour model introduced as a strong monad. This provides a basis to reason about (and transform) state-based software designs.

Keywords: components, refinement, coalgebraic models.

1 Introduction

Component-based software development [15, 16] emerged as a promising paradigm to deal with the ever increasing need for mastering complexity, software evolution and reuse. From object-orientation it retains the basic principle of encapsulation of data and code. The emphasis, however, is shifted from (class) inheritance to (object) composition to avoid interference between the former and encapsulation and, thus, paving the way to a development methodology based on *third-party assembly* of components. In [3, 2], the authors proposed a coalgebraic characterization of software components as specifications of *state-based* modules, encapsulating a number of services through a public interface and providing limited access to an internal state space. Component persist and evolve in time, being able to interact with the environment during their overall computation. This piece of research has been driven by two key ideas: first, the 'black-box' characterization of software components favors an *observational* semantics; secondly, the proposed constructions should be *generic* in the sense that they should not depend on a particular notion of component behaviour. This led both to the adoption of coalgebra theory [14] to capture observational semantics and to the abstract characterization of possible behaviour models (*e.g.*, partiality or (different degrees of) non-determinism) by strong monads acting as parameters in the resulting calculus.

Within this approach, briefly reviewed in section 2, a set of component connectors have been identified and their properties established as *bisimilarity* equations with respect to a generic behaviour model. Actually, the corner stone of

our 'components as coalgebras' approach is the use of coinduction to prove $\sim$-results, where $\sim$ is the appropriate bisimilarity relation, as a basis for reasoning and transforming component-based designs. This paper provides a basis to extend the approach toward the *inequational* side through the discussion of suitable notions of *refinement*.

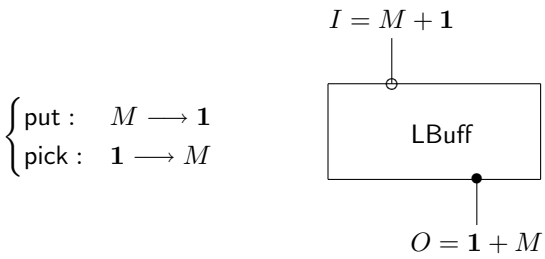
In broad terms *refinement* can be defined as a *transformation* of an 'abstract' into a more 'concrete' design, entailing a notion of *substitution*, but not necessarily *equivalence*. There is, however, a diversity of ways of understanding both what *substitution* means, and what such a *transformation* should seek for. In *data refinement*, for example, after Hoare's landmark paper [8], the 'concrete' model is required to have *enough redundancy* to represent all the elements of the 'abstract' one. This is captured by the definition of a surjection from the former into the latter (the *retrieve map*). Also *substitution* is regarded as 'complete' in the sense that the (concrete) operations accept all the input values accepted by the corresponding abstract ones, and, for the same inputs, the results produced are also the same, up to the retrieve map. This means that, if models are specified, as they usually are in *model-oriented* design methods like VDM[10], in terms of pre and post-conditions, the former are weakened and the latter strengthened, under refinement. In *object-orientation*, on the other hand, *substitution* is expressed in terms of *behaviour subtyping* [11] capturing the idea that 'concrete' objects behave similarly to objects in the 'abstract' class. Finally, refinement in *process algebras* is usually discussed in terms of several 'observation' preorders (see, for example, [7]), most of them justifying transformations entailing *reduction of nondeterminism*.

In general, refinement correctness means that the usage of a system according to its 'abstract' description is still valid if it is actually built according to the 'concrete' one. What is commonly understood by being a *valid usage* is that the corresponding observable consequences are still the same, or, in a less strict sense, a subset thereof. The exact definition, however, depends on the underlying *behaviour model*, which, in our approach to component modelling, is taken as a specification parameter. Therefore, the main contribution of this paper is a semantic characterization of refinement for state-based components, parametric on a strong monad intended to capture components' behavioural models.

After a brief review of the component calculus, in section 2, the paper discusses two levels of component refinement: the *interface* level, concerned with what one may call *plugging compatibility*, in section 3, and the *behavioural* one in section 4, which introduces *forward* and *backward* morphisms as refinement 'witnesses', and section 5 which builds on them to propose a family of refinement preorders. Section 6 proves soundness of *simulations* to establish behavioural refinement. A few examples, along with some prospects for future work, are presented in section 7.

2 Components as Coalgebras

In [3, 2] software components and connectors have been characterised as dynamic systems with a public interface and a private, encapsulated state. A typical example is LBuff: a connector modelling a buffered channel which occasionally looses messages, as represented below:



The put and pick operations are regarded as 'buttons' or 'ports', whose signatures are grouped together in the diagram (M stands for a message parameter type, $\mathbf{1}$ for the nullary datatype and $+$ for 'datatype sum'). One might capture LBuff dynamics by a function $a_{\mathsf{LBuff}} : U \times I \longrightarrow \mathcal{P}(U \times O)$ where U denotes the space state. This describes how LBuff reacts to input stimuli, produces output data (if any) and changes state. It can also be written in a curried form as $\overline{a}_{\mathsf{LBuff}} : U \longrightarrow \mathcal{P}(U \times O)^I$ that is, as a *coalgebra* [14] of signature $U \longrightarrow \mathsf{T}\, U$ where functor T captures the transition 'shape':

$$\mathsf{T} = \mathcal{P}(\mathsf{Id} \times O)^I \tag{1}$$

Built in this 'shape' is the possibility of non deterministic evolution captured by the use of $\mathcal{P}$, the finite powerset monad. Concretely, LBuff is defined over $U = M^*$, with nil as the initial state, and dynamics given by

$$\begin{aligned} a_{\mathsf{LBuff}}\langle u, \mathsf{put}\ m\rangle &= \{\langle u, \iota_1\ *\rangle, \langle m : u, \iota_1\ *\rangle\} \\ a_{\mathsf{LBuff}}\langle u, \mathsf{pick}\rangle &= \{\langle \mathsf{tail}\ u, \iota_2\ (\mathsf{head}\ u)\rangle\} \end{aligned}$$

where put m and pick abbreviates $\iota_1\ m$ and $\iota_2\ *$, respectively.

Non determinism, capturing the occasional loss of messages, is a possible behavioural pattern for this buffer, but, by no means, the only one. Other components will exhibit different *behaviour models*: actually *genericity* is achieved by replacing the *powerset* monad above, by an arbitrary *strong monad*[3] B. In the general case, a component $p : I \longrightarrow O$ is specified as a (pointed) coalgebra in Set

$$\langle u_p \in U_p, \overline{a}_p : U_p \longrightarrow \mathsf{B}(U_p \times O)^I \rangle \tag{2}$$

[3] A *strong monad* is a monad $\langle \mathsf{B}, \eta, \mu\rangle$ where B is a strong functor and both η and μ strong natural transformations. B being strong means there exist natural transformations $\tau_r^{\mathsf{T}} : \mathsf{T} \times - \Longrightarrow \mathsf{T}(\mathsf{Id} \times -)$ and $\tau_l^{\mathsf{T}} : - \times \mathsf{T} \Longrightarrow \mathsf{T}(- \times \mathsf{Id})$ called the right and left strength, respectively, subject to certain conditions. Their effect is to *distribute* the free variable values in the context "$-$" along functor B.

where point u_p is taken as the 'initial' or 'seed' state. Therefore, the computation of an action will not simply produce an output and a continuation state, but a B-structure of such pairs. The monadic structure provides tools to handle such computations. Unit (η) and multiplication (μ), provide, respectively, a value embedding and a 'flatten' operation to reduce nested behavioural annotations. Strength, either in its right (τ_r) or left (τ_l) version, will cater for context information.

In such a framework, components become *arrows* in a (bicategorical) universe Cp whose objects are sets, which provide types to input/output parameters (the components' *interfaces*), and component morphisms $h : p \longrightarrow q$ are functions relating the state spaces of p and q and satisfying the following *seed preservation* and *coalgebra* conditions:

$$h\ u_p\ =\ u_q \qquad \text{and} \qquad \overline{a}_q \cdot h\ =\ \mathsf{B}\ (h \times O)^I \cdot \overline{a}_p \tag{3}$$

For each triple of objects $\langle I, K, O\rangle$, a composition law is given by functor $\mathbf{;}_{I,K,O} : \mathsf{Cp}(I,K) \times \mathsf{Cp}(K,O) \longrightarrow \mathsf{Cp}(I,O)$ whose action on objects p and q is

$$p\,\mathbf{;}\,q\ =\ \langle\langle u_p, u_q\rangle \in U_p \times U_q, \overline{a}_{p;q}\rangle \qquad \text{with}$$

$$\begin{aligned} a_{p;q}\ = U_p \times U_q \times I\ &\xrightarrow{\cong}\ U_p \times I \times U_q\ \xrightarrow{a_p \times \mathsf{id}}\ \mathsf{B}(U_p \times K) \times U_q \\ &\xrightarrow{\tau_r}\ \mathsf{B}(U_p \times K \times U_q)\ \xrightarrow{\cong}\ \mathsf{B}(U_p \times (U_q \times K)) \\ &\xrightarrow{\mathsf{B}(\mathsf{id} \times a_q)}\ \mathsf{B}(U_p \times \mathsf{B}(U_q \times O))\ \xrightarrow{\mathsf{B}\tau_l}\ \mathsf{BB}(U_p \times (U_q \times O)) \\ &\xrightarrow{\cong}\ \mathsf{BB}(U_p \times U_q \times O)\ \xrightarrow{\mu}\ \mathsf{B}(U_p \times U_q \times O) \end{aligned}$$

Similarly, for each object K, an identity law is given by a functor $\mathsf{copy}_K : \mathbf{1} \longrightarrow \mathsf{Cp}(K,K)$ whose action is the constant component $\langle * \in \mathbf{1}, \eta_{\mathbf{1}\times K}\rangle$. Note that the definitions above rely solely on the monadic structure of B.

In [3, 2] a set of component *combinators* have been defined upon Cp in a similar parametric way and their properties studied. In particular it was shown that any function $f : A \longrightarrow B$ can be lifted to Cp as $\ulcorner f \urcorner = \langle * \in \mathbf{1}, \eta_{(\mathbf{1}\times B)} \cdot (\mathsf{id} \times f)\rangle$. Also defined were both a *wrapping* mechanism $p[f,g]$ which encodes the pre- and post-composition of a component with Cp-lifted functions, and three tensors, capturing, respectively, *external choice* ($\boxplus : I + J \longrightarrow O + R$), *parallel* ($\boxtimes : I \times J \longrightarrow O \times R$) and *concurrent* ($\circledast : I + J + I \times J \longrightarrow O + R + O \times R$) composition, given components $p : I \longrightarrow O$ and $q : J \longrightarrow R$. When interacting with $p \boxplus q : I + J \longrightarrow O + R$, the environment chooses either to input a value of type I or one of type J, which triggers the corresponding component (p or q, respectively), producing the relevant output. In its turn, *parallel* composition corresponds to a synchronous product: both components are executed simultaneously when triggered by a pair of legal input values. Note, however, that the behaviour effect, captured by monad B, propagates. For example, if B can express component failure and one of the arguments fails, the product will fail as well. Finally, *concurrent* composition combines choice and parallel, in the sense that p and q can be executed independently or jointly, depending on input. Generalized interaction is catered through a 'feedback' mechanism on a subset of the inputs.

3 Interface Refinement

Component *interface refinement* is concerned with type compatibility. The question is whether a component can be transformed, by suitable wiring, to replace another component with a different interface. As the structure of components interface types encodes the available operations, this may capture situations of *extension of functionality*, in the sense that the 'concrete' component may introduce new operations. In the context of object-orientation, this is often called *design sophistication* (rather than *refinement*) and it is known not to be a congruence with respect to typical process combinators (see *e.g.*, [17]). If we structure component input and output parameters as an operations' signature, interface refinement can also be seen as induced by a signature morphism, as in *e.g.*, [13].

To motivate our own approach, consider, from [3], the following law expressing commutativity of *choice*:

$$p \boxplus q \ \sim \ (q \boxplus p)[\mathsf{s}_+, \mathsf{s}_+] \tag{4}$$

where $\mathsf{s}_+ : I + J \longrightarrow J + I$ is a natural isomorphism capturing $+$ commutativity. The law states that $p \boxplus q$ and $q \boxplus p$ are bisimilar *up to isomorphic wiring*. This means that the observational effect of component $p \boxplus q$ can be achieved by $q \boxplus p$, providing the interface of the latter is converted to the interface of the former. Such a conversion is achieved by composition with the appropriate *wires*, leading to a notion of *replaceability*.

Definition 1 *Let p and q be components. We say that $p : I \longrightarrow O$ is* replaceable *by $q : I' \longrightarrow O'$, or q is a* replacement *of p, and write $p \lessdot q$ if there exist functions $w_1 : I \longrightarrow I'$ and $w_2 : O' \longrightarrow O$, to be referred to as the replacement* witnesses*, such that*

$$p \ \sim \ q[w_1, w_2] \tag{5}$$

Furthermore, components p and q are interchangeable *if each of them is a replacement of the other. Formally,*

$$p \doteqdot q \quad \textit{iff} \quad p \lessdot q \ \wedge \ q \lessdot p \tag{6}$$

Clearly, $p \boxplus q \doteqdot q \boxplus p$, using isomorphism s_+ as a wire in both cases. In general, $p \doteqdot q$ whenever w_1 and w_2 in (5) are isomorphisms.

Lemma 1. *Replaceability ($\lessdot$) is a preorder on components*

Proof. Clearly, $\lessdot$ is reflexive because $p \lessdot p$ is witnessed by $p \sim p[\mathsf{id}, \mathsf{id}]$. On the other hand, if $p \lessdot q$ and $q \lessdot r$ hold, there exist w_1, w_2, w_3 and w_4 such that $p \sim q[w_1, w_2]$ and $q \sim r[w_3, w_4]$. Thus, a composition result on wrapping [2] and transitivity of $\sim$, entails $p \sim r[w_1 \cdot w_3, w_4 \cdot w_2]$, *i.e.*, $p \lessdot r$.

□

Using $\lessdot$ and $\div$, some component laws in [2], as (4) above, can be presented in a 'wiring free' form. As another example consider the law relating *concurrent* composition with *choice*,

$$\ulcorner \iota_1 \urcorner ; (p \circledast q) \sim (p \boxplus q) ; \ulcorner \iota_1 \urcorner$$

which gives rise to two replacement inequations:

$$\ulcorner \iota_1 \urcorner ; (p \circledast q) \lessdot p \boxplus q \quad \text{and} \quad (p \boxplus q) ; \ulcorner \iota_1 \urcorner \lessdot p \circledast q$$

Finally, the statement that every component p can replace an *inert* component can be expressed as an interface refinement situation: $\mathsf{inert} \lessdot p$.

Relation $\lessdot$, however, fails to be a pre-congruence with respect to the component operators introduced in [3]. It is easy to check that $\boxplus$, $\boxtimes$ and $\circledast$, as well as *wrapping* are preserved, *i.e.*, if $p \lessdot p'$ then, for any q, f and g, $p[f, g] \lessdot p'[f, g]$, $p \boxplus q \lessdot p' \boxplus q$ and, similarly, for the other two tensors. But things are different with respect to sequential composition and feedback. In these cases, the replaced expression may even become wrongly typed.

What $p \lessdot p'$ means is that component p can be replaced in *any* context by $p'[w_1, w_2]$, for any functions w_1, w_2 witnessing the fact. The explicit reference to them is actually required, something which is not completely satisfactory in a refinement situation, although common in similar settings (*cf.* [17]).

4 Forward and Backward Morphisms

Interface refinement is essentially concerned with plugging adjustment. Behaviour refinement, on the other hand, affects the internal dynamics of a component while leaving unchanged its external interface: it takes place inside each hom-category of Cp. Intuitively component p is a *behavioural refinement* of q if the behaviour patterns observed from p are a structural restriction, with respect to the *behavioural model* captured by monad B, of those of q. To make precise such a 'definition' we shall first describe behaviour patterns concretely as *generalized transitions*.

Actually, just as transition systems can be coded back as coalgebras, any coalgebra $\langle U, p : U \longrightarrow \mathsf{T}U \rangle$ specifies a (T-shaped) transition structure over its carrier U. For extended polynomial Set endofunctors[4] such a structure may be expressed as a binary relation $\longrightarrow_p \subseteq U \times U$, defined in terms of the *structural membership* relation $\in_{\mathsf{T}} \subseteq U \times \mathsf{T}\, U$, *i.e.*,

$$u \longrightarrow_p u' \quad \mathsf{iff} \quad u' \in_{\mathsf{T}} p\, u$$

[4] The class inductively defined as the least collection of functors containing the identity Id and constant functors $\underline{K}$ for all objects K in the category, closed by functor composition and finite application of product, coproduct, covariant exponential and finite powerset functors.

where $\in_{\mathsf{T}}$ is defined by induction of the structure of T:

$$\begin{array}{rcl} x \in_{\mathsf{Id}} y & \text{iff} & x = y \\ x \in_{\underline{K}} y & \text{iff} & \mathsf{false} \\ x \in_{\mathsf{T}_1 \times \mathsf{T}_2} y & \text{iff} & x \in_{\mathsf{T}_1} \pi_1\ y\ \vee\ x \in_{\mathsf{T}_2} \pi_2\ y \\ x \in_{\mathsf{T}_1 + \mathsf{T}_2} y & \text{iff} & \begin{cases} y = \iota_1\ y' & \Rightarrow x \in_{\mathsf{T}_1} y' \\ y = \iota_2\ y' & \Rightarrow x \in_{\mathsf{T}_2} y' \end{cases} \\ x \in_{\mathsf{T}^{\underline{K}}} y & \text{iff} & \exists_{k \in K}.\ x \in_{\mathsf{T}} y\ k \\ x \in_{\mathcal{P}\mathsf{T}} y & \text{iff} & \exists_{y' \in y}.\ x \in_{\mathsf{T}} y' \end{array}$$

Notice that, given $x \in U$, $X \in \mathsf{T}U$ and a function $h : U \longrightarrow V$, if $x \in_{\mathsf{T}} X$ then $h\ x \in_{\mathsf{T}} \mathsf{T}h\ X$, as it may be shown by induction on the polynomial structure, resorting to the definition of $\in_{\mathsf{T}}$ and functoriality. Similarly, the dynamics of $p : I \longrightarrow O$, based on functor $\mathsf{B}(\mathsf{Id} \times O)^I$, can be expressed in terms of the following transition relation:

$$u \xrightarrow{\langle i,o \rangle}_p u' \quad \text{iff} \quad \langle u', o \rangle \in_{\mathsf{B}} (pu)\ i$$

In this setting, a possible (and intuitive) way of regarding component p as a behavioural refinement of q is to consider that p transitions are simply *preserved* in q. For non deterministic components this is understood simply as set inclusion. But one may also want to consider additional restrictions. For example, to stipulate that if p has no transitions from a given state, q should also have no transitions from the corresponding state(s). Or one may adopt the dual point of view requiring transition *reflection* instead of preservation. In any case the same basic question arises: how can such a refinement stituation be identified?

In data refinement, as mentioned above, there is a 'recipe' to identify a refinement situation: look for a *retrieve function* to witness it. I.e., a morphism in the relevant category, from the 'concrete' to the 'abstract' model such that the latter can be *recovered* from the former up to a suitable notion of equivalence, though, typically, not in a unique way.

In our components' framework, however, things do not work this way. The reason is obvious: component morphisms are (seed preserving) coalgebra morphisms which are known (*e.g.*, [14]) to entail bisimilarity. Therefore we have to look for a somewhat *weaker* notion of a morphism between coalgebras.

First of all recall that a component morphism from p to q is a seed preserving function $h : U_p \longrightarrow U_q$ such that

$$\mathsf{B}(h \times \mathsf{id}) \cdot a_p\ =\ a_q \cdot (h \times id) \tag{7}$$

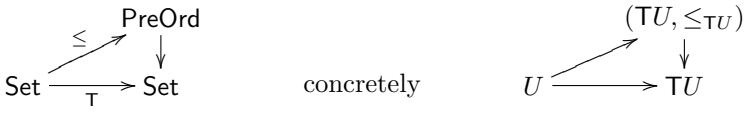
In terms of transitions, equation (7) is translated into the following two requirements (by a straightforward generalization of an argument in [14]):

$$u \xrightarrow{\langle i,o \rangle}_p u' \ \Rightarrow\ h\ u \xrightarrow{\langle i,o \rangle}_q h\ u' \tag{8}$$

$$h\ u \xrightarrow{\langle i,o \rangle}_q v' \ \Rightarrow\ \exists_{u' \in U}.\ u \xrightarrow{\langle i,o \rangle}_p u'\ \wedge\ v' = h\ u' \tag{9}$$

which jointly state that, not only p dynamics, as represented by the induced transition relation, is *preserved* by h (8), but also q dynamics is *reflected* back over the same h (9). Is it possible to weaken the morphism definition to capture only one of these aspects? The answer is given as follows:

An order $\leq$ on a Set endofunctor T is defined in [9] as a functor $\leq$ which makes the following diagram to commute:



This means that for any function $h : X \longrightarrow Y$, $\mathsf{T}h$ preserves the order, *i.e.*

$$x_1 \leq_{\mathsf{T}X} x_2 \;\Rightarrow\; (\mathsf{T}h)\; x_1 \leq_{\mathsf{T}Y} (\mathsf{T}h)\; x_2 \tag{10}$$

In the sequel $\leq$ will be referred to as a *refinement preorder*. Then,

Definition 2 *Let* T *be an extended polynomial functor on* Set *and consider two* T*-coalgebras* $\alpha : U \longrightarrow \mathsf{T}U$ *and* $\beta : V \longrightarrow \mathsf{T}V$*. A* forward *morphism* $h : \alpha \longrightarrow \beta$ *with respect to a refinement preorder* $\leq$*, is a function from* U *to* V *such that*

$$\mathsf{T}\, h \cdot \alpha \;\leq\; \beta \cdot h$$

Dually, h *is called a* backwards *morphism if*

$$\beta \cdot h \;\leq\; \mathsf{T}\, h \cdot \alpha$$

The following lemma shows that such morphisms compose and can be taken as witnesses of refinement situations:

Lemma 2. *For* T *an endofunctor in* Set*,* T*-coalgebras and forward (respectively, backward) morphisms define a category.*

Proof. In both cases, identities are the identities on the carrier and composition is inherited from Set. What remains to be shown is that the composition of forward (respectively, backward) morphisms yields again a forward (respectively, backward) morphism. So, let $h : \alpha \longrightarrow \beta$ and $k : \beta \longrightarrow \gamma$ be two forward (respectively, backward) morphisms. Then

(*forward* case)

$$
\begin{array}{ll}
 & \mathsf{T}(k \cdot h) \cdot \alpha \\
= & \quad \{\ \mathsf{T} \text{ functor } \} \\
 & \mathsf{T}k \cdot (\mathsf{T}h \cdot \alpha) \\
\leq & \quad \{\ h \text{ forward and (10)} \ \} \\
 & \mathsf{T}k \cdot (\beta \cdot h) \\
= & \quad \{\ \cdot \text{ associative } \} \\
 & (\mathsf{T}k \cdot \beta) \cdot h \\
\leq & \quad \{\ k \text{ forward } \} \\
 & (\gamma \cdot k) \cdot h \\
= & \quad \{\ \cdot \text{ associative } \} \\
 & \gamma \cdot (k \cdot h)
\end{array}
$$

(*backward* case)

$$
\begin{array}{ll}
 & \gamma \cdot (k \cdot h) \\
= & \quad \{\ \cdot \text{ associative } \} \\
 & (\gamma \cdot k) \cdot h \\
\leq & \quad \{\ k \text{ backward } \} \\
 & (\mathsf{T}k \cdot \beta) \cdot h \\
= & \quad \{\ \cdot \text{ associative } \} \\
 & \mathsf{T}k \cdot (\beta \cdot h) \\
\leq & \quad \{\ h \text{ backward and (10)} \ \} \\
 & \mathsf{T}k \cdot \mathsf{T}h \cdot \alpha \\
= & \quad \{\ \mathsf{T} \text{ functor } \} \\
 & \mathsf{T}(k \cdot h) \cdot \alpha
\end{array}
$$

□

Such a split of a coalgebra morphism, witnessing a bisimulation equation, into two conditions, makes it possible to capture separately transition *preservation* and *reflection*. To prove the next result, however, it is necessary to impose an extra condition on the refinement preorder $\leq$ expressing its compatibility with $\in_{\mathsf{T}}$: for all $x \in X$ and $x_1, x_2 \in \mathsf{T}X$,

$$x \in_{\mathsf{T}} x_1 \ \wedge \ x_1 \leq x_2 \ \Rightarrow \ x \in_{\mathsf{T}} x_2 \tag{11}$$

Lemma 3. *Let* T *be an extended polynomial functor in* Set*, and* α *and* β *two* T*-coalgebras as above. Let* $\longrightarrow_\alpha$ *and* $\longrightarrow_\beta$ *denote the corresponding transition relations. A forward (respectively, backward) morphism* $h : \alpha \longrightarrow \beta$ *preserves (respectively, reflects) such transition relations.*

Proof. Preservation follows from

$$
\begin{array}{ll}
 & u \longrightarrow_\alpha u' \\
\equiv & \quad \{\ \longrightarrow \text{ definition } \} \\
 & u' \in_{\mathsf{T}} \alpha\ u \\
\Rightarrow & \quad \{\ \in_{\mathsf{T}} \text{ definition } \} \\
 & h\ u' \in_{\mathsf{T}} (\mathsf{T}h \cdot \alpha)\ u \\
\equiv & \quad \{\ h \text{ forward and (11)} \ \} \\
 & h\ u' \in_{\mathsf{T}} (\beta \cdot h)\ u \\
\equiv & \quad \{\ \cdot \text{ associative and } \longrightarrow \text{ definition } \} \\
 & h\ u \longrightarrow_\beta h\ u'
\end{array}
$$

To establish reflection suppose that $h\ u \longrightarrow_\beta v'$, *i.e.*, $v' \in_{\mathsf{T}} (\beta \cdot h)\ u$. As h is a backward morphism we have $\beta \cdot h \leq \mathsf{T}\ h \cdot \alpha$, which, together with requirement (11), entails $v' \in_{\mathsf{T}} (\mathsf{T}\ h \cdot \alpha)\ u$. This implies the existence of a state $u' \in U$ such that $v' = h\ u'$ and $u' \in_{\mathsf{T}} \alpha\ u$, *i.e.*, $u \longrightarrow_\alpha u'$.

□

5 Behaviour Refinement

The existence of a *forward* (*backward*) morphism connecting two components p and q witnesses a refinement situation whose symmetric closure coincides, as expected, with bisimulation. In the sequel we will restrict ourselves to *forward* refinement[5] and define *behaviour refinement* as the existence of a forward morphism *up to bisimulation*. Formally,

Definition 3 *Component p is a behaviour refinement of q, written $q \trianglelefteq p$, if there exist components r and s such that $p \sim r$, $q \sim s$ and a (seed preserving) forward morphism from r to s.*

The exact meaning of a refinement assertion $q \trianglelefteq p$ depends, of course, on the concrete refinement preorder $\leq$ adopted. Let us consider a few possibilities.

- T-*structural* inclusion, defined by $x \leq y$ iff $\forall_{e \in_{\mathsf{T}x}}.\ e \in_{\mathsf{T}} y$, seems inadequate because the transition relation preserved by a forward morphism is not $\xrightarrow{\langle i,o \rangle}_p$, but simply $\longrightarrow_p$, and, therefore, blind to the outputs produced. This suggests an additional requirement on refinement preorders for Cp components: their definition on a constant functor $\underline{K}$ is equality on set K, *i.e.*, $x \leq_{\underline{K}} y$ iff $x =_K y$ so that transitions with different O-labels could not be related.
- Building on this idea, we arrive to a first (good) example:

$$
\begin{array}{rcl}
x \subseteq_{\mathsf{Id}} y & \text{iff} & x = y \\
x \subseteq_{\underline{K}} y & \text{iff} & x =_K y \\
x \subseteq_{\mathsf{T}_1 \times \mathsf{T}_2} y & \text{iff} & \pi_1\ x \subseteq_{\mathsf{T}_1} \pi_1\ y\ \wedge\ \pi_2\ x \subseteq_{\mathsf{T}_2} \pi_2\ y \\
x \subseteq_{\mathsf{T}_1 + \mathsf{T}_2} y & \text{iff} & \begin{cases} x = \iota_1\ x' \wedge y = \iota_1\ y' & \Rightarrow x' \subseteq_{\mathsf{T}_1} y' \\ x = \iota_2\ x' \wedge y = \iota_2\ y' & \Rightarrow x' \subseteq_{\mathsf{T}_2} y' \end{cases} \\
x \subseteq_{\mathsf{T}^{\underline{K}}} y & \text{iff} & \forall_{k \in K}.\ x\ k \subseteq_{\mathsf{T}} y\ k \\
x \subseteq_{\mathcal{P}\mathsf{T}} y & \text{iff} & \forall_{e \in x} \exists_{e' \in y}.\ e \subseteq_{\mathsf{T}} e'
\end{array}
$$

  A *forward* refinement of non deterministic components based on $\subseteq_{\mathsf{T}}$ captures the classical notion of *nondeterminism reduction*.

[5] A similar study can be made about *backward* refinement, although the underlying intuition seems less familiar.

– However, this preorder can be tuned to more specific cases. For example, the following 'failure forcing' variant — $\subseteq^E_{\mathsf{T}}$, where E stands for 'emptyset' —- guarantees that the concrete component fails no more than the abstract one. It is defined as $\subseteq_{\mathsf{T}}$ by replacing the clause for the powerset functor by

$$x \subseteq^E_{\mathcal{P}\mathsf{T}} y \quad \mathsf{iff} \quad (x = \emptyset \Rightarrow y = \emptyset) \ \wedge \ \forall_{e \in x} \exists_{e' \in y}.\ e \subseteq_{\mathsf{T}} e'$$

– Relation $\subseteq_{\mathsf{T}}$ is inadequate for partial components: refinement would collapse into bisimilarity, instead of entailing increasing definition in the implementation. An alternative is relation $\subseteq^F_{\mathsf{T}}$ (F standing for 'failure') which replaces the sum clause in $\subseteq_{\mathsf{T}}$ by

$$x \subseteq^F_{\mathsf{T}_1+\mathsf{T}_2} y \quad \mathsf{iff} \quad \begin{cases} x = \iota_1\ x' \wedge y = \iota_1\ y' & \Rightarrow x' \subseteq_{\mathsf{T}} y' \\ x = \iota_2\ * & \Rightarrow y = \iota_2\ * \\ y = \iota_2\ * & \Rightarrow \mathsf{true} \end{cases}$$

To illustrate behaviour refinement, consider the lossy buffer LBuff introduced in section 2, and a deterministic buffered channel Buff specified as a coalgebra $M^* \longrightarrow (M^* \times (\mathbf{1} + M))^{M+\mathbf{1}}$ with nil as the initial state, and dynamics given by

$$\begin{aligned} a_{\mathsf{Buff}}\langle u, \mathsf{put}\ m\rangle &= \langle m : u, \iota_1\ *\rangle \\ a_{\mathsf{Buff}}\langle u, \mathsf{pick}\rangle &= \langle \mathsf{tail}\ u, \iota_2\ (\mathsf{head}\ u)\rangle \end{aligned}$$

To establish $\mathsf{LBuff} \trianglelefteq \mathsf{Buff}$ it is required first to embed the latter into the space of non-deterministic systems. This is achieved by a (natural) transformation from $(\mathsf{Id} \times O)^I$ to $\mathcal{P}(\mathsf{Id} \times O)^I$ canonically extending function $\mathsf{sing}\, x = \{x\}$ which is a monad morphism from the *identity* to the *powerset* monads — the behaviour models underlying Buff and LBuff, respectively. Then, it is immediate to verify that the identity function on state space M^* is a *forward* morphism, with respect to the first preorder given above, *i.e.*,

$$(\mathsf{id} \times O) \cdot a_{\mathsf{Buff}} \ \subseteq \ a_{\mathsf{LBuff}}$$

Another behaviour refinements of LBuff would arise by choosing different strategies for delivering elements from the buffer. Here are some possibilities, each of them is witnessed by a forward morphism:

– the queuing strategy, leading to the specification Buff as above;
– the stack strategy (LIFO deliver);
– the priority strategy (in which elements carry some probability information);
– the lift strategy (a linear order on the elements is served in alternating increasing and decreasing order).

In the priority strategy, for example, elements are labelled with a 'show-up' probability, introducing an elementary form of probabilistic nondeterminism. As detailed in [3], the corresponding behaviour monad is generated by a monoid $M = \langle [0,1], \mathsf{min}, \times\rangle$ with the additional requirement that for each $m \in M$, $\sum(\mathcal{P}\pi_2)m = 1$. 'Probabilistic' components can be embedded into the space of 'plain nondeterministic' ones where behaviour refinement, wrt $\subseteq_{\mathsf{T}}$, is discussed.

6 Simulations

In this section we prove that behaviour refinement, as characterized above, can be established by a *simulation* relation $R \subseteq U_p \times U_q$ on the state spaces of the 'concrete' (p) and the 'abstract' (q) components. Again, the notion of a simulation depends on the adopted refinement preorder $\leq$. To proceed in a *generic* way, we adopt an equally generic definition of simulation due to Jacobs and Hughes in [9]:

Definition 4 *Given a* Set *endofunctor* T *and a refinement preorder* $\leq$*, a lax relation lifting is an operation* $Rel_{\leq}(\mathsf{T})$ *mapping relation* R *to* $\leq \circ Rel(\mathsf{T})(R) \circ \leq$*, where* $Rel(\mathsf{T})(R)$ *is the lifting of* R *to* T *(defined, as usual, as the* T*-image of inclusion* $\langle r_1, r_2\rangle : R \longrightarrow U \times V$*,* i.e.*,* $\langle \mathsf{T}r_1, \mathsf{T}r_2\rangle : \mathsf{T}R \longrightarrow \mathsf{T}U \times \mathsf{T}V$*).*

Given T*-coalgebras* α *and* β*, a simulation is a* $Rel_{\leq}(\mathsf{T})$*-coalgebra over* α *and* β*,* i.e.*, a relation* R *such that, for all* $u \in U, v \in V$*,* $\langle u, v\rangle \in R \Rightarrow \langle \alpha\, u, \beta\, v\rangle \in Rel_{\leq}(\mathsf{T})(R)$*.*

In order to prove that simulations are a sound proof technique to establish behaviour refinement we consider separately functional and non functional simulations. In any case, however, simulations are assumed to be left total relations [6] as we do not consider *partial* refinements.

Lemma 4. *Let* p *and* q *be* T*-components over state spaces* U *and* V*, respectively. For a given refinement preorder* $\leq$*, if there exists a simulation* $R \subseteq U \times V$ *which is both functional and left total, then* p *is a (forward) refinement of* q*.*

Proof. By assumption, simulation R is the graph of a function. Now, define a forward morphism $h : U \to V$ as $h\, u = v$ iff $\langle u, v\rangle \in R$. Because R is a simulation, for every pair $\langle u, v\rangle \in R$, there should exist $x \in \mathsf{T}U$, $y \in \mathsf{T}V$, such that $\alpha\, u \leq_{\mathsf{T}U} x$, $y \leq_{\mathsf{T}V} \beta\, v$, and $\langle x, y\rangle \in Rel(\mathsf{T})(R)$, i.e., $y = \mathsf{T}h(x)$. By (10) and $\alpha\, u \leq_{\mathsf{T}U} x$ we get $\mathsf{T}h(\alpha\, u) \leq_{\mathsf{T}V} \mathsf{T}h(x)$, and thus $\mathsf{T}h(\alpha\, u) \leq_{\mathsf{T}V} \beta\, v$. Since R is left total, h is defined for all $u \in U$, making the following diagram to commute:

$$\begin{array}{ccc} u & \xrightarrow{\quad h \quad} & h\, u = v \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\beta} \\ \alpha\, u\, (\leq_{\mathsf{T}U} \alpha\, u) & \xrightarrow{\mathsf{T}h} & \mathsf{T}h(\alpha\, u) \leq_{\mathsf{T}V} \quad \beta\, v \end{array}$$

□

Consider, now, the non-functional case (*e.g.* whenever two bisimilar but not equal abstract states are represented by a single concrete state). To prove soundness in this case, the state space of the 'concrete' component p is artificially inflated with an auxiliary state space such that a forward morphism can be found.

[6] A relation $R \subseteq U \times V$ is *functional* if every $u \in U$ is related to at most one $v \in V$ and *left total* if for all $u \in U$, there exists some $v \in V$ such that $\langle u, v\rangle \in R$.

Definition 5 *Given a coalgebra $(U, \alpha : U \to \mathsf{T}U)$ and a set W, define the extension of α to W as any coalgebra $\widehat{\alpha}$ over $\widehat{U} = U \times W$ such that $\mathsf{T}\pi_1 \circ \widehat{\alpha} = \alpha \circ \pi_1$.*

Clearly this auxiliary state space does not interfere with the behaviour of α: π_1 being a coalgebra morphism, the two coalgebras are bisimilar.

Given components p and q and a non-functional simulation R an auxiliary coalgebra $\widehat{p}$ can be defined taking R as the state space (which, because R is left total, is just an extension of p in the sense of the definition above) and the rule $(u', v') \in_{\mathsf{T}} \widehat{\alpha}(u, v)$ iff $u' \in_{\mathsf{T}} a_p u \wedge v' \in_{\mathsf{T}} a_q v$ as its dynamics. With this construction we prove that

Lemma 5. *(Soundness) To prove $q \trianglelefteq p$ it is sufficient to exhibit a left total simulation R relating components p and q.*

Proof. If R is functional the result follows from lemma 4. Otherwise construct $\widehat{p}$ as above: clearly p is bisimilar to $\widehat{p}$ and the graph of projection π_2 from its state space to V defines a simulation between $\widehat{p}$ and q. By definition, $p \sim \widehat{p}$ and the existence of a (seed-preserving) forward morphism from $\widehat{p}$ to q entails $q \trianglelefteq p$.

□

Finally notice that, although $\trianglelefteq$ is transitive, it is not always the case that simulations are closed under (relational) composition. This would be a consequence of $Rel_{\leq}(\mathsf{T})$ preserving composition, but, in general, only the following weaker result holds:

Lemma 6. *Any refinement preorder $\leq$ verifies*

$$Rel_{\leq}(\mathsf{T})(R \circ S) \subseteq Rel_{\leq}(\mathsf{T})(R) \circ Rel_{\leq}(\mathsf{T})(S) \quad \textit{and} \quad =_{\mathsf{T}U} \subseteq Rel_{\leq}(\mathsf{T})(=_U)$$

Proof. For the first statement note that $\langle u, w\rangle \in Rel_{\leq}(\mathsf{T})(R \circ S)$ equivales

$$\begin{aligned}
&\exists u', w'.(u \leq u' \wedge \langle u', w'\rangle \in Rel(\mathsf{T})(R \circ S) \wedge w' \leq w)\\
&\{\text{because } Rel(\mathsf{T})(R \circ S) = Rel(\mathsf{T})(R) \circ Rel(\mathsf{T})(S)\}\\
\Leftrightarrow&\exists u', w'.(u \leq u' \wedge (\exists v'.(\langle u', v'\rangle \in Rel(\mathsf{T})(R) \wedge \langle v', w'\rangle \in Rel(\mathsf{T})(S))) \wedge w' \leq w)\\
\Leftrightarrow&\exists u', w', v'.(u \leq u' \wedge \langle u', v'\rangle \in Rel(\mathsf{T})(R) \wedge \langle v', w'\rangle \in Rel(\mathsf{T})(S) \wedge w' \leq w)\\
&\{introducing\ v = v'\}\\
\Rightarrow&\exists u', w', v, v'.(u \leq u' \wedge \langle u', v'\rangle \in Rel(\mathsf{T})(R) \wedge v' \leq v) \wedge\\
&(v \leq v' \wedge \langle v', w'\rangle \in Rel(\mathsf{T})(S) \wedge w' \leq w)\\
\Rightarrow&\exists v.\langle u, v\rangle \in Rel_{\leq}(\mathsf{T})(R) \wedge (v, w) \in Rel_{\leq}(\mathsf{T})(S)\\
\Leftrightarrow&\langle u, w\rangle \in Rel_{\leq}(\mathsf{T})(R) \circ Rel_{\leq}(\mathsf{T})(S)
\end{aligned}$$

Then consider

$$\begin{aligned}
=_{\mathsf{T}U} \ &\subseteq \leq_{\mathsf{T}U}\\
&= \leq_{\mathsf{T}U} \circ =_{\mathsf{T}U} \circ \leq_{\mathsf{T}U} \ = \ \leq_{\mathsf{T}U} \circ Rel(\mathsf{T})(=_U) \circ \leq_{\mathsf{T}U} \ = \ Rel_{\leq}(\mathsf{T})(=_U)
\end{aligned}$$

□

7 Discussion and Future Work

In this paper, two levels of refinement for (state-based) components have been introduced. In particular, the notion of *behavioural* refinement parametric on a model of behaviour captured by a strong monad B is, to the best of our knowledge, new. It is *generic* enough to capture a number of situations, depending on both B and the refinement preorder adopted. Nondeterminism reduction is just one possibility among many others. Also note that Poll's notion of *behavioural subtyping* in [13], at the model level, emerges as a particular instantiation.

As mentioned in the introduction, the main motivation underlying this work is the development of *inequational* laws in the context of the component calculus proposed in [3]. Even though there is not enough space in this paper to introduce the derived laws, let us take a brief glimpse. As a first example consider equation

$$\ulcorner !_I \urcorner \sim p \,;\, \ulcorner !_O \urcorner \tag{12}$$

which does not hold for non trivial behaviour models. In fact the Cp lifting of the final arrow (as the lifting of any other function) cannot fail, whereas the the right hand side may fail (whenever p does). Function $! : U_p \times \mathbf{1} \longrightarrow \mathbf{1}$ is, however, a forward morphism, with respect to $\subseteq^F_{\mathsf{T}}$ for partial components, or to both $\subseteq_{\mathsf{T}}$ and $\subseteq^E_{\mathsf{T}}$ for non deterministic ones. For this last case, note that $\overline{a}_{\ulcorner !_O \urcorner} \cdot ! = \lambda\, i \in I.\ \{*\}$, whereas $\mathsf{B}(! \times \mathsf{id})^I \cdot \overline{a}_{p;\ulcorner !_O \urcorner}\ \langle u, * \rangle$ equals

$$\lambda\, i \in I\ .\ \begin{cases} \{*\} & \mathsf{iff}\ (\overline{a}_p\, u)\,(i) \neq \emptyset \\ \emptyset & \mathsf{iff}\ (\overline{a}_p\, u)\,(i) = \emptyset \end{cases}$$

Therefore, $\ulcorner !_I \urcorner \trianglelefteq p \,;\, \ulcorner !_O \urcorner$. Similarly, the *cancellation* law for parallel composition $\boxtimes$, which involves a *split*-like construction for components (which, differently from the split of functions [4], is not an universal arrow), is, in general, a refinement result:

$$p \trianglelefteq \langle p, q \rangle \,;\, \ulcorner \pi_1 \urcorner \tag{13}$$

witnessed by projection $\pi_1 : U_p \times U_q \times 1 \to U_p$ as a forward morphism. Yet another example is given by the (pseudo) naturality of $\ulcorner \triangle \urcorner$, where $\triangle$ is the diagonal function, which could be written as

$$\ulcorner \triangle \urcorner \,;\, (p \boxtimes p) \trianglelefteq p \,;\, \ulcorner \triangle \urcorner \tag{14}$$

Finally, *monotonicity* of $\trianglelefteq$ with respect to both *pipeline* composition and the tensor products can be proved by defining a simulation in terms of the argument simulations: if $q \trianglelefteq p$ and $t \trianglelefteq r$ are witnessed by R_1 and R_2, respectively, refinement $q \,\square\, t \trianglelefteq p \,\square\, r$, with $\square$ standing for $;, \boxtimes, \boxplus$ or $\circledast$ is witnessed by simulation $R = \{((u_p, u_r), (u_q, u_t)) \mid (u_p, u_q) \in R_1 \wedge (u_r, u_t) \in R_2\}$.

Currently we are working on the full development of the refinement calculus and, in particular, in its application to the proof of consistency between static and dynamic diagrams in UML in the context of [12]. Whether this approach scales up to be useful in the classification and transformation of software *architectures* [1] remains a research question. Further comparison with refinement

theories in both process algebra (as in, *e.g.*, [5]) and state-based systems (for example in [6]) is also in order.

Acknowledgements. This piece of research was carried on in the context of the PURe Project (*Program Understanding and Re-engineering*) supported by Fct (the Portuguese Foundation for Science and Technology) under contract `POSI/ICHS/44304/2002`. The work of Sun Meng was further supported by the National Natural Science Foundation of China under grant no. 60273001.

References

1. R. Allen and D. Garlan. A formal basis for architectural connection. *ACM TOSEM*, 6(3):213–249, 1997.
2. L. S. Barbosa. Towards a Calculus of State-based Software Components. *Journal of Universal Computer Science*, 9(8):891–909, August 2003.
3. L. S. Barbosa and J. N. Oliveira. State-based components made generic. In H. P. Gumm, editor, *CMCS'03, Elect. Notes in Theor. Comp. Sci.*, volume 82.1, 2003.
4. R. Bird and O. Moor. *The Algebra of Programming.* Series in Computer Science. Prentice-Hall International, 1997.
5. P. Degano, R. Gorrieri, and G. Rosolini. A categorical view of process refinement. In J. de Bakker, G. Rozenberg, and J. Rutten, editors, *Proc. REX Workshop on Semantics*, pages 138–154. Springer Lect. Notes Comp. Sci. (666), 1992.
6. J. Derrick and E. Boiten. Calculating upward and downward simulations of state-based specifications. *Information and Software Technology*, 41:917–923, July 1999.
7. M. Fokkinga and R. Eshuis. Comparing refinements for failure and bisimulation semantics. Technical report, Faculty of Computing Science, Enschede, 2000.
8. C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.
9. B. Jacobs and J. Hughes. Simulations in coalgebra. In H. P. Gumm, editor, *CMCS'03, Elect. Notes in Theor. Comp. Sci.*, volume 82.1, Warsaw, April 2003.
10. C. B. Jones. *Systematic Software Development Using* Vdm. Series in Computer Science. Prentice-Hall International, 1986.
11. B. Liskov. Data abstraction and hierarchy. *SIGPLAN Notices*, 23(3), 1988.
12. S. Meng and B. Aichernig. Towards a Coalgebraic Semantics of UML: Class Diagrams and Use Cases. Technical Report 272, UNU/IIST, January 2003.
13. E. Poll. A coalgebraic semantics of subtyping. *Theorectical Informatica and Aplications*, 35(1):61–82, 2001.
14. J. Rutten. Universal coalgebra: A theory of systems. *Theor. Comp. Sci.*, 249(1):3–80, 2000. (Revised version of CWI Techn. Rep. CS-R9652, 1996).
15. C. Szyperski. *Component Software, Beyond Object-Oriented Programming.* Addison-Wesley, 1998.
16. P. Wadler and K. Weihe. Component-based programming under different paradigms. Technical report, Dagstuhl Seminar 99081, February 1999.
17. J. Woodcock and J. Davies. *Using* Z*: Specification, Refinement and Proof.* Prentice-Hall International, 1996.