Journal of Universal Computer Science, vol. 9, no. 8 (2003), 891-909
submitted: 24/2/03, accepted: 30/5/03, appeared: 28/8/03

Towards a Calculus of State-based Software Components

Luís S. Barbosa
(Dep. Informática, Universidade do Minho, Portugal
lsb@di.uminho.pt)

Abstract: This paper introduces a calculus of state-based software components modelled as concrete coalgebras for some Set endofunctors, with specified initial conditions. The calculus is parametrized by a notion of *behaviour*, introduced as a strong (usually commutative) monad. The proposed component model and calculus are illustrated through the characterisation of a particular class of components, classified as *separable*, which includes the ones arising in the so-called *model oriented* approach to systems' design.
Key Words: software components, coalgebra, semantics
Category: D.3.1, F.3.2

1 Introduction

The emerging *component-orientation* paradigm [Wadler and Weihe, 1999] retains from object-orientated programming the basic principle of encapsulation of data and code, but shifts the emphasis 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 [Szyperski, 1998]. The paradigm is often illustrated by the visual metaphor of a *palette* of computational units, treated as black boxes, and a *canvas* into which they can be dropped. Connections are established by drawing *wires*, corresponding to some sort of interfacing code. The expression *software component*, however, is semantically overloaded, standing, in fact, for a collection of technologies and a research concern common to different communities [1]. Moreover, component-orientation has grown up to a popular technology before consensual definitions and principles, let alone formal foundations, have been put forward. In this paper we limit ourselves to regard 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. Components persist and evolve in time, being able to interact with the environment during their overall computation.

Our starting point is the conjunction of two key ideas: first, the 'black-box' characterisation of software components favours an *observational* semantics; secondly, the proposed constructions should be *generic* in the sense that they should

[1] As put by P. Wadler in a 1999 Seminar suggestively entitled 'Component-based Programming under different paradigms', *Just as Eskimos need fifty words for ice, perhaps we need many words for components.*

not depend on a particular notion of component behaviour. Observational semantics is nicely captured by coalgebra theory [Rutten, 2000]. A coalgebra for an (endo)functor T is a map $p : U \longrightarrow \mathsf{T}\ U$ which may be thought of as a transition structure, of *shape* T, on a set U (the *state space*). The shape of T describes not only the way the state is (partially) accessed through the *observers*, but also how it evolves through *actions*. The lack of constructors forces equality to be replaced by *bisimilarity* and induction by *coinduction* as a proof principle[2]. The other key idea is the application of the so-called *functorial* approach to datatypes, originated in the work of the ADJ group in the early seventies [Goguen et al., 1977], to the area of state-based systems modelling. This approach provides a basis for *generic programming* [Backhouse et al., 1998] which raises the level of abstraction of the programming discourse in a way such that seemingly disparate techniques and algorithms are unified into idealised, kernel programming schemata.

In such a context this paper builds on previous work by the author (see *e.g.*, [Barbosa and Oliveira, 2003, Barbosa and Oliveira, 2002]) on coalgebraic modeling. Both the component model and the corresponding calculus discussed in the sequel are parametrized by a notion of behaviour, introduced as a strong monad, to express, for example, partiality or (different degrees of) non-determinism. The paper is organised as follows: sections 2 and 3 introduce the basic component's combinators and their calculus. This is tunned to a particular, but common, class of components — the *separable* ones — in section 4, which introduces the relevant interaction combinators. A (toy) application is discussed in section 5. Finally, section 6 points out some connections to related research and future work.

2 Components As Coalgebras

Let I and O be sets acting as component interfaces. A component p with input I and output O, represented as $p : I \longrightarrow O$, is modelled as a *concrete*, seeded coalgebra for the following Set endofunctor:

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

where B is a strong monad abstracting from each specific behaviour model[3]. I.e., a pair $\langle u_p \in U_p, \overline{a}_p : U_p \longrightarrow \mathsf{B}(U_p \times O)^I \rangle$, where a specific value u_p of the

[2] Recall that, given two coalgebras p and q, a *comorphism* from p to q is a function h from the carrier of p to that of q which preserves the coalgebra dynamics, *i.e.*, such that $\mathsf{T}h \cdot p = q \cdot h$ holds. From any coalgebra there is one and only one comorphism to the *final* coalgebra, if existent. Therefore, the *final* coalgebra collects all possible behaviours up to bisimilarity, in the same sense that an initial algebra collects all terms up to isomorphism. From a programming point of view, it is usually referred to as a *coinductive* type.

[3] Such a parametrization is a source of *genericity* in the approach; see appendix A for a quick overview and [Barbosa and Oliveira, 2003] for a detailed discussion.

state space U_p is taken as the coalgebra 'initial' or 'seed' value and the dynamics of the latter is captured by currying a state-transition function $a_p : U_p \times I \longrightarrow \mathsf{B}\,(U_p \times O)$. In this way, 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.

In such a framework, components 'are arrows' and so arrows between components are 'arrows between arrows', which motivates the adoption of a *bicategorical* [Benabou, 1967] framework to structure our reasoning universe. In particular, a *2-cell* $h : p \longrightarrow q$ is a function relating the state spaces of p and q and satisfying the following *seed preservation* and *coalgebra* conditions:

$$h\; u_p \;=\; u_q \qquad \text{and} \qquad \overline{a}_q \cdot h \;=\; \mathsf{T}^{\mathsf{B}}\; h \cdot \overline{a}_p \tag{2}$$

2-cell composition is inherited from Set and the identity $1_p : p \longrightarrow p$, on component p, is defined as the identity id_{U_p} on the carrier of p.
Let us denote by Cp such a bicategory. For each triple of objects $\langle I, K, O\rangle$, a composition law is given by a functor $;_{I,K,O} : \mathsf{Cp}(I,K) \times \mathsf{Cp}(K,O) \longrightarrow \mathsf{Cp}(I,O)$ whose action on objects p and q is given by

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

where $a_{p;q} : U_p \times U_q \times I \longrightarrow \mathsf{B}(U_p \times U_q \times O)$ is detailed as follows

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

The action of ';' on 2-cells is simply given by $h\,;k \;=\; h \times k$. Also notice that the definition above relies solely on properties of the monad morphisms η, μ, strengths τ_r and τ_l, and the distributive law δ_l[4]. Finally, for each object K, an identity law is given by a functor

$$\mathsf{copy}_K : \mathbf{1} \longrightarrow \mathsf{Cp}(K,K)$$

whose action on objects is the constant component $\langle * \in \mathbf{1}, \overline{a}_{\mathsf{copy}_K}\rangle$, where $a_{\mathsf{copy}_K} = \eta_{\mathbf{1}\times K}$. Similarly, the action on morphisms is the constant comorphism $\mathsf{id}_{\mathbf{1}}$.

The fact that, for each strong monad B, components form a bicategory [5] amounts not only to a standard definition of the two basic combinators (';' and copy_K) of the component calculus, but also to setting up its basic laws (equations

[4] Such monad morphisms as well as common 'housekeeping' functions are recalled in appendix A.

[5] The reader is referred to [Barbosa, 2001] for all omitted proofs.

(9) and (10) in appendix B). Such laws are stated as T^{B}-bisimilarity equations, *i.e.*, up to observation through the 'shape' encoded in functor T^{B}.

Any function $f : A \longrightarrow B$ can be regarded as a (particular case of a) component, being lifted to Cp as

$$\ulcorner f \urcorner \; = \; \langle * \in \mathbf{1}, \overline{a}_{\ulcorner f \urcorner} \rangle$$

i.e., as a coalgebra over $\mathbf{1}$ whose action is given by the currying of

$$a_{\ulcorner f \urcorner} = \mathbf{1} \times A \xrightarrow{\mathsf{id} \times f} \mathbf{1} \times B \xrightarrow{\eta_{(\mathbf{1} \times B)}} \mathsf{B}(\mathbf{1} \times B)$$

Clearly, function lifting is functorial (laws (11) and (12) in appendix B). Moreover, isomorphisms, split monos and split epis lift to Cp as, respectively, isomorphisms, split monos and split epis [Barbosa, 2001].

The coalgebraic specification of a component describes immediate reactions to possible state/input configurations. It is its extension in time which gives the component's *behaviour* (abstracted as an element of the *final* coalgebra). Formally, the behaviour $[\![p]\!]$ of a component p is computed by *coinductive extension* [Rutten, 2000], *i.e.*, $[\![p]\!] = [\![(\overline{a}_p)]\!] u_p$, where $[\![p]\!]$ is the unique function from p to the final coalgebra.

Behaviours organise themselves in a category Bh whose objects are sets and each arrow $b : I \longrightarrow O$ is an element of $\nu_{I,O}$, the carrier of the final coalgebra $\omega_{I,O}$ for functor $\mathsf{B}(\mathsf{Id} \times O)^I$. Note that the structure of Bh mirrors whatever structure Cp possesses. In fact, the former is isomorphic to a sub-(bi)category of the latter whose arrows are components defined over the corresponding final coalgebra. Alternatively, we may think of Bh as constructed by quotienting Cp by the greatest T^{B}-bisimulation. However, as final coalgebras are fully abstract with respect to bisimulation, the bicategorical structure collapses: the hom-categories become simply hom-*sets*. Of course, properties holding in Cp up to bisimulation, do hold 'on the nose' in the behaviour category.

3 Component Combinators

Components can be aggregated in a number of different ways, besides 'pipeline' composition discussed above. This section explores the structure of Cp_{B} by introducing a 'wrapping' combinator, which may be thought of as an extension of the *renaming* connective found in process calculi (*e.g.*, [Milner, 1989]), and three tensors. Generalized interaction, through a sort of 'feedback' mechanism is addressed in the next section. Note the definition of all combinators is *parametric* on the *behaviour model*, relying on generic properties of the strong monad B. Component combinators are shown to be either lax endofunctors in Cp_{B} or simply functors between families of hom-categories. Their definitions carry naturally to Bh_{B}, where they show up as behaviour connectives, defining a particular (typed) 'process' algebra.

Wrapping

The pre- and post-composition of a component with Cp-lifted functions can be encapsulated in an unique *wrapping* combinator. Let $p : I \longrightarrow O$ be a component and consider functions $f : I' \longrightarrow I$ and $g : O \longrightarrow O'$. By $p[f, g]$ we will denote 'component p wrapped by f and g'. This has type $I' \longrightarrow O'$ and is defined by input pre-composition with f and output post-composition with g. Formally, the wrapping combinator is a functor (between the corresponding hom-categories) $-[f, g] : \mathsf{Cp}(I, O) \longrightarrow \mathsf{Cp}(I', O')$ which is the identity on morphisms and maps a component $\langle u_p, \overline{a}_p\rangle$ into $\langle u_p, \overline{a}_{p[f,g]}\rangle$, where

$$a_{p[f,g]} = U_p \times I' \xrightarrow{\mathsf{id}\times f} U_p \times I \xrightarrow{a_p} \mathsf{B}(U_p \times O) \xrightarrow{\mathsf{B}(\mathsf{id}\times g)} \mathsf{B}(U_p \times O')$$

Equations (14) and (15) in appendix B capture wrapping basic properties.

Some simple components arise by lifting elementary functions to Cp. We have already remarked that the lifting of the canonical arrow associated to the initial Set object plays the role of an *inert* component, unable to react to the outside world. Let us give this component a name:

$$\mathsf{inert}_A = \ulcorner ?_A \urcorner \tag{3}$$

In particular, we define the nil component

$$\mathsf{nil} = \mathsf{inert}_\emptyset = \ulcorner ?_\emptyset \urcorner = \ulcorner \mathsf{id}_\emptyset \urcorner \tag{4}$$

typed as $\mathsf{nil} : \emptyset \longrightarrow \emptyset$. Note that any component $p : I \longrightarrow O$ can be made inert by wrapping. For example, $p[?_I, !_O] \sim \mathsf{inert}_{\mathbf{1}}$.

A somewhat dual role is played by component $\mathsf{idle} = \ulcorner \mathsf{id}_{\mathbf{1}} \urcorner$. Note that $\mathsf{idle} : \mathbf{1} \longrightarrow \mathbf{1}$ is always willing to propagate an unstructured stimulus (*e.g.*, the push of a button) leading to a (similarly) unstructured reaction (*e.g.*, exciting a led).

Tensors

Three tensor products are introduced to capture three different composition patterns: *external choice* ($p \boxplus q$), *parallel* ($p \boxtimes q$) and *concurrent* composition ($p \circledast q$).

Let $p : I \longrightarrow O$ and $q : J \longrightarrow R$ be two components defined by $\langle u_p, \overline{a}_p\rangle$ and $\langle u_q, \overline{a}_q\rangle$, respectively. When interacting with $p \boxplus q : I + J \longrightarrow O + R$, the environment will be allowed to choose either to input a value of type I or one of type J, which will trigger the corresponding component (p or q, respectively), producing the relevant output. On its turn, *parallel* composition yields $p \boxtimes q : I \times J \longrightarrow O \times R$, corresponding 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 the input supplied.

The three combinators are defined as lax functors from $\mathsf{Cp} \times \mathsf{Cp}$ to Cp. Choice, for example, consists of an action on objects given by $I \boxplus J \;=\; I + J$ and a family of functors $\boxplus_{I,O,J,R} : \mathsf{Cp}(I,O) \times \mathsf{Cp}(J,R) \longrightarrow \mathsf{Cp}(I+J, O+R)$ yielding

$$\boxplus_{I,O,J,R} : \mathsf{Cp}(I,O) \times \mathsf{Cp}(J,R) \longrightarrow \mathsf{Cp}(I+J, O+R)$$

yielding

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

where

$$\begin{aligned} a_{p \boxplus q} \;=\;\; & U_p \times U_q \times (I+J) \xrightarrow{\mathsf{dr}} U_p \times U_q \times I + U_p \times U_q \times J \\ & \xrightarrow{\mathsf{xr}+\mathsf{a}} U_p \times I \times U_q + U_p \times (U_q \times J) \\ & \xrightarrow{a_p \times \mathsf{id} + \mathsf{id} \times a_q} \mathsf{B}\;(U_p \times O) \times U_q + U_p \times \mathsf{B}\;(U_q \times R) \\ & \xrightarrow{\tau_r + \tau_l} \mathsf{B}\;(U_p \times O \times U_q) + \mathsf{B}\;(U_p \times (U_q \times R)) \\ & \xrightarrow{\mathsf{Bxr}+\mathsf{Ba}^\circ} \mathsf{B}\;(U_p \times U_q \times O) + \mathsf{B}\;(U_p \times U_q \times R) \\ & \xrightarrow{[\mathsf{B}\;(\mathsf{id} \times \iota_1), \mathsf{B}\;(\mathsf{id} \times \iota_2)]} \mathsf{B}\;(U_p \times U_q \times (O+R)) \end{aligned}$$

which maps pairs of arrows $\langle h_1, h_2 \rangle$ into $h_1 \times h_2$. Parallel composition is similarly defined: just take $I \boxtimes J$ as $I \times J$ and

$$\begin{aligned} a_{p \boxtimes q} \;=\;\; & U_p \times U_q \times (I \times J) \xrightarrow{\mathsf{m}} U_p \times I \times (U_q \times J) \\ & \xrightarrow{a_p \times a_q} \mathsf{B}\;(U_p \times O) \times \mathsf{B}\;(U_q \times R) \\ & \xrightarrow{\delta_l} \mathsf{B}\;(U_p \times O \times (U_q \times R)) \\ & \xrightarrow{\mathsf{B\;m}} \mathsf{B}\;(U_p \times U_q \times (O \times R)) \end{aligned}$$

Finally, regarding $\circledast$ also as a lax functor, define $I \circledast J = I + J + I \times J$, and $a_{p \circledast q} \;=$

$$\begin{aligned} & U_p \times U_q \times (I \circledast J) \xrightarrow{\mathsf{dr}} U_p \times U_q \times (I+J) + U_p \times U_q \times (I \times J) \\ & \xrightarrow{a_{p \boxplus q} + a_{p \boxtimes q}} \mathsf{B}\;(U_p \times U_q \times (O+R)) + \mathsf{B}\;(U_p \times U_q \times (O \times R)) \\ & \xrightarrow{[\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)]} \mathsf{B}\;(U_p \times U_q \times (O \circledast R)) \end{aligned}$$

Properties of these combinators are established in detail in [Barbosa, 2001]. Besides the verification of their definition as lax functors, it is shown that all of them are symmetric tensor products (commutativity of $\boxtimes$ and $\circledast$ depending on monad B being Abelian). In particular, nil defined as $\ulcorner \mathsf{id}_\emptyset \urcorner$, is the unit of both $\boxplus$ and $\circledast$ and a zero element for $\boxtimes$. The unit for parallel composition is $\mathsf{idle} = \ulcorner \mathsf{id}_\mathbf{1} \urcorner$. Laws (18) and (32) relate $\boxplus$ and $\boxtimes$ with the usual Set sum and product, respectively, whereas laws (50) and (51) show them as suitable specialisations of $\circledast$.

Seeking for Universals

A fundamental question arising in the development of a components' calculus is whether typical *universal* constructions used in (data-oriented) program calculi (see *e.g.*, [Bird and Moor, 1997]) have a counterpart. Clearly, for any set I, the lifting of $?_I : \emptyset \longrightarrow I$ to Cp keeps naturality, *i.e.*, $\ulcorner ?_I \urcorner \, ; p \sim \ulcorner ?_O \urcorner$. As any bisimulation equation lifts to an equality in the behaviours category, $\emptyset$ is *initial* in Bh. For no trivial B, however, functions to $\mathbf{1}$ loose their naturality once lifted and, therefore, Bh has no final object.

Consider now combinators $\boxplus$ and $\boxtimes$. Is it possible to define counterparts in Cp to the *either* and *split* constructions in Set, by making[6]

$$[p, q] = (p \boxplus q) \, ; \ulcorner \nabla \urcorner \quad \text{and} \quad \langle p, q \rangle = \ulcorner \triangle \urcorner \, ; (p \boxtimes q) \; ?$$

The answer is only partially positive. In fact, codiagonal ∇ does not keep naturality once lifted to Cp, even for $\mathsf{B} = \mathsf{Id}$. Hence, uniqueness of component $[p, q]$ (see diagram on appendix B) is lost. However, *cancellation*, *reflection* and *absorption* laws still hold in Cp (equations (23), (24) and (25) in appendix B, respectively) and, therefore, $\boxplus$ becomes a weak coproduct in Bh. Also note that Set coproduct embeddings — once lifted to Cp, — keep their naturality (equations (27) and (28) in appendix B), paving the way to the derivation of an 'idempotency' result (equation (29) in appendix B).

The dual situation, involving $\boxtimes$ and *split*, is a bit different. The problem here is that a *cancellation* result — $\langle p, q \rangle \, ; \ulcorner \pi_1 \urcorner \sim p$ — is only valid for a monad B which excludes the possibility of *failure* (*e.g.*, the non-empty powerset). On the other hand, diagonal $\triangle$ keeps its naturality when lifted to Cp, for B expressing deterministic behaviour (*e.g.*, the identity or the *maybe* monad), entailing a *fusion* law:

$$r \, ; \langle p, q \rangle \sim \langle r \, ; p, r \, ; q \rangle$$

Combining these two results, one concludes that $\boxtimes$ is a *product* in Bh, but only for behaviour models excluding failure and no determinism, which narrows the applicability scope of this fact to the category of total deterministic components. However, *reflection* and *absorption* laws (equations (37) and (38) in appendix B) do hold for any B.

[6] where $\triangle = \langle \mathsf{id}, id \rangle : I \longrightarrow I \times I$ and $\nabla = [\mathsf{id}, id] : I + I \longrightarrow I$ are, respectively, the diagonal and codiagonal functions.

4 Separable Components and Interaction

So far component interaction has been centred upon sequential composition, which is the Cp counterpart to functional composition in Set. This basic interaction scheme can be generalised by introducing *partial* connections, *i.e.*, by connecting some input to some output wires and, consequently, forcing *part* of the output of a component to be fed back as input. Some general feedback mechanisms have been studied in [Barbosa, 2001]. In this paper, however, we shall restrict ourselves to a particular one which, being quite powerful, can only be defined over a sub-category of Cp — that of *separable* components.

A separable component is specified as a collection of actions over a shared state space, each of which exhibits its own input and output types. Packing them into a T^{B}-coalgebra, results into a component with an *additive* interface such that each type of input stimulus produces a result whose type is known and unique (among the possible results). Such components arise typically in the practical use of model oriented specification methods, such as VDM [Jones, 1986] or Z [Spivey, 1992]. In the sequel a *restriction* combinator and an interaction scheme, called *hook*, will be defined and some of their properties investigated.

To be precise, let us call *separable* a component $p : I + J \longrightarrow O + R$ whose dynamics $\overline{a}_p$ can be split into two independent coalgebras

$$\overline{a}_{1.p} \ : U_p \longrightarrow \mathsf{B}(U_p \times O)^I \quad \text{and} \quad \overline{a}_{2.p} \ : U_p \longrightarrow \mathsf{B}(U_p \times R)^J$$

This notation suggests that $1.p$ and $2.p$ can also be regarded as independent components, with different interfaces, but defined over the same state space U_p and seed value u_p. Therefore the dynamics of p arises as the currying of

$$a_p \ = \ [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \mathsf{dr}$$

Clearly, for each behaviour monad B and objects I, J, O and R, separable components form a subcategory $\mathsf{Cp}(I + J, O + R)_{Sp}$ of $\mathsf{Cp}(I + J, O + R)$. Moreover, Cp-arrows connecting separable components are characterised by the following property: given $p, q : I + J \longrightarrow O + R$ separable, h is a morphism from p to q iff the same h, seen as an arrow in the underlying category, is also a comorphism from $1.p$ to $1.q$ and $2.p$ to $2.q$ (with a slight abuse of notation we shall write $h : 1.p \longrightarrow 1.q$ and $h : 2.p \longrightarrow 2.q$). The proof follows:

Proof. To prove the right to left implication assume $h \ : \ 1.p \ \longrightarrow \ 1.q$ and $h \ :$

$2.p \longrightarrow 2.q$. Then

$$
\begin{array}{ll}
& \mathsf{B}(h \times \mathsf{id}) \cdot a_p \\
= & \quad \{ \ p \text{ separable } \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.p}] \cdot \mathsf{dr} \\
= & \quad \{ \ + \text{ fusion, identity and } + \text{ absorption } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (\mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p} + \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p}) \cdot \mathsf{dr} \\
= & \quad \{ \ \text{assumption: } h : 1.p \longrightarrow 1.q \text{ and } h : 2.p \longrightarrow 2.q \ \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{1.q} \cdot (h \times \mathsf{id}) + a_{2.q} \cdot (h \times \mathsf{id})) \cdot \mathsf{dr} \\
= & \quad \{ \ \mathsf{dr} \text{ natural } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{1.q} + a_{2.q}) \cdot \mathsf{dr} \cdot (h \times \mathsf{id}) \\
= & \quad \{ \ q \text{ separable } \} \\
& a_q \cdot (h \times \mathsf{id})
\end{array}
$$

For the reverse implication, assume $h : p \longrightarrow q$. Then,

$$
\begin{array}{ll}
& h : p \longrightarrow q \\
\equiv & \quad \{ \ \text{comorphism condition and } p, q \text{ separable } \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{1.p} + a_{2.p}) \cdot \mathsf{dr} \\
& \ = \ [\mathsf{B}(\mathsf{id} \times \iota_1), \mathsf{B}(\mathsf{id} \times \iota_2)] \cdot (a_{1.q} + a_{2.q}) \cdot \mathsf{dr} \cdot (h \times \mathsf{id}) \\
\equiv & \quad \{ \ + \text{ fusion, } + \text{ absorption and } \mathsf{dr} \text{ natural } \} \\
& [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p}, \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p}] \cdot \mathsf{dr} \\
& \ = \ [\mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.q} \cdot (h \times \mathsf{id}), \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.q} \cdot (h \times \mathsf{id})] \cdot \mathsf{dr} \\
\equiv & \quad \{ \ \text{equality } \} \\
& \mathsf{B}(\mathsf{id} \times \iota_1) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p} \ = \ \mathsf{B}(\mathsf{id} \times \iota_1) \cdot a_{1.q} \cdot (h \times \mathsf{id}) \\
& \text{and} \\
& \mathsf{B}(\mathsf{id} \times \iota_2) \cdot \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p} \ = \ \mathsf{B}(\mathsf{id} \times \iota_2) \cdot a_{2.q} \cdot (h \times \mathsf{id}) \\
= & \quad \{ \ \star \ \} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p} \ = \ a_{1.q} \cdot (h \times \mathsf{id}) \\
& \text{and} \\
& \mathsf{B}(h \times \mathsf{id}) \cdot a_{2.p} \ = \ a_{2.q} \cdot (h \times \mathsf{id}) \\
= & \quad \{ \ \text{comorphism condition} \} \\
& h : 1.p \longrightarrow 1.q \ \text{ and } \ h : 2.p \longrightarrow 2.q
\end{array}
$$

where the $\star$ step is justified as follows. If $I \neq \emptyset$, ι_1 is a split mono and so is $\mathsf{B}(\mathsf{id} \times \iota_1)$, because any functor preserves split monos, and we are done. If $I = \emptyset$, the result holds trivially as both $\mathsf{B}(h \times \mathsf{id}) \cdot a_{1.p}$ and $a_{1.q} \cdot (h \times \mathsf{id})$ have source

$U_p \times \emptyset$ and any function from such a domain to an arbitrary set X can be written as $?_X \cdot \mathsf{zr}_{U_p}$.

$\square$

Separable components support a number of specific combinators — we shall concentrate here in feedback schemes. First of all, consider $p : I + Z \longrightarrow Z + O$ separable. Then, the feed back of a Z output will always produce a O one. Thus, a feedback combinator, which actually restricts the component interface, may be defined as

$$\big(p\big)_Z : I \longrightarrow O \;=\; \langle u_p \in U_p, \overline{a}_{(p)_Z} \rangle$$

$$a_{(p)_Z} \;=\; U_p \times I \xrightarrow{a_{2.p} \bullet a_{1.p}} \mathsf{B}(U_p \times O)$$

where $\bullet$ denotes the Kleisli composition for monad B. A particular case of a separable component is $q \boxplus r : I + Z \longrightarrow Z + O$, where $q : I \longrightarrow Z$ and $r : Z \longrightarrow O$. Clearly, $\big(q \boxplus r\big)_Z \sim q \,;\, r$.

A more general situation arises whenever p has type $p : I + J + Z \longrightarrow Z + O + R$. and is separable into three 'threads' $1.p$, $2.p$ and $3.p$, with $1.p$ and $3.p$ composable. Then, for each I, Z, J, R and O, the *hook* combinator is defined as a family of functors $\big(p\big)_Z : \mathsf{Cp}(I{+}J{+}Z, Z{+}O{+}R)_{Sp} \longrightarrow \mathsf{Cp}(I{+}J, O{+}R)_{Sp}$ which, being the identity on arrows, map each component $p : I + J + Z \longrightarrow Z + O + R$ to

$$\big(p\big)_Z : I + J \longrightarrow O + R \;=\; \langle u_p \in U_p, \overline{a}_{(p)_Z} \rangle$$

where

$$\begin{aligned} a_{(p)_Z} \;=\; & U_p \times (I + J) \xrightarrow{\mathsf{dr}} U_p \times I + U_p \times J \\ & \xrightarrow{(a_{3.p} \bullet a_{1.p}) + a_{2.p}} \mathsf{B}(U_p \times R) + \mathsf{B}(U_p \times O) \\ & \xrightarrow{[\mathsf{B}(\mathsf{id} \times \iota_2), \mathsf{B}(\mathsf{id} \times \iota_1)]} \mathsf{B}(U_p \times (O + R)) \end{aligned}$$

Clearly, for each I, Z, J, R and O, the *hook* combinator is a functor from $\mathsf{Cp}(I + J + Z, Z + O + R)_{Sp}$ to $\mathsf{Cp}(I + J, O + R)_{Sp}$ [7].

We shall conclude our digression around separable components by stating some properties of the interaction between *hook* and other component combinators. Proof arguments rely on the separability of the composites.

The first law relates *hook* with wrapping: *hook* is well behaved with respect to a 'structural' wrapping function, if its component in the feed back parameter is

[7] Note that this definition subsumes the previous one: by isomorphic wiring the interface of $p : I + Z \longrightarrow Z + O$ can be re-written as $I + \emptyset + O \longrightarrow Z + \emptyset + O$ which is separable, with the original p threads in the first and third positions and $?_{\mathsf{B}(U_p \times \emptyset)}$ in the second.

an isomorphism. Thus, let $p : I + J + Z \longrightarrow Z + O + R$ be a separable component and $i : W \longrightarrow Z$ an isomorphism. Then,

$$\big(p[f + g + i, i^\circ + h + k]\big)_W \;\sim\; \big(p\big)_Z[f + g, h + k] \tag{5}$$

The *hook* combinator can be thought of as a *partial* sequential composition, in the sense that joined 'pins' vanish from the outermost interface. This gives rise to a number of bisimilar aggregation schemes, based on either **;** or $\boxplus$, that allow the specifier to play around with the components involved. For example, it can be shown that

$$\big((p \boxplus r)\,[\mathsf{xr}_+, \mathsf{id}]\big)_Z \;\sim\; \big(p\big)_Z \boxplus r \tag{6}$$

for $p : I + Z \longrightarrow Z + O$ and $r : J \longrightarrow R$. On the other hand, a number of laws make it possible to swap a partial composition from the input to the output of the feed back 'pin'. The law below is prototypical of this class of situations. Let $p : I + J + Z \longrightarrow W + O + R$ be separable and $s : W \longrightarrow Z$. Then,

$$\big((\mathsf{copy}_I \boxplus \mathsf{copy}_J \boxplus s)\,;p\big)_W \;\sim\; \big(p\,;(s \boxplus \mathsf{copy}_O \boxplus \mathsf{copy}_R)\big)_Z \tag{7}$$

Similarly, for $p : I + Z \longrightarrow O + R$, $r : J \longrightarrow W$ and $s : W \longrightarrow Z$,

$$\big(((r\,;s) \boxplus p)\,[\mathsf{a}_+, \mathsf{a}_+]\big)_Z \;\sim\; \big((r \boxplus ((\mathsf{copy}_I \boxplus s)\,;p))\,[\mathsf{a}_+, \mathsf{a}_+]\big)_W \tag{8}$$

The proof idea for (7) is that, because p is separable, component s is activated in both expressions only once and always triggered by a p response to an I typed input. A lengthy, but trivial, expression manipulation establishes $\mathsf{s} \cdot (\mathsf{s} \times \mathsf{id}) : \mathbf{1} \times \mathbf{1} \times U_s \times U_p \longrightarrow U_p \times (U_s \times \mathbf{1} \times \mathbf{1})$ as a comorphism from the left to the right hand side of the equation.

5 Applying the Calculus

To illustrate the component calculus in action consider a voting system in which stimuli sent by independent voting pads are counted in a central unit (the 'concentrator') until a certain level is reached. A common use of such a system can be found in processing units for electronic opinion polls, as in some television shows. A similar system, however, can be used to count inputs from a number of sensors in, *e.g.*, an industrial plant. Typically, such sensors emit a number of stimuli before terminating. In any case, the *maybe* monad is an adequate choice for the behaviour model. The voting system is built around two basic components: the *voting pad* VP and the *concentrator* C as detailed below. The voting pad maintains, as state information, the number of stimuli remaining to be emitted. Its interface is elementary: trivial input and the output means that no special data is exchanged by this component: simply, a button is pushed (on input) and a led excited (on output). Thus, $\mathsf{VP} : \mathbf{1} \longrightarrow \mathbf{1} \quad = \quad \langle n \in \mathbb{N}, \overline{a}_{\mathsf{VP}} \rangle$ where[8]

$$a_{\mathsf{VP}}\ \langle n, * \rangle\ = (n \neq 0\ \rightarrow\ \iota_1\ \langle n-1, * \rangle,\ \iota_2\ *)$$

On the other hand, the concentrator is modelled by component $\mathsf{C} : \mathbb{N} + \mathbf{1} \longrightarrow \mathbf{1} + \mathbf{2} \quad = \quad \langle 0 \in \mathbb{N}, \overline{a}_{\mathsf{C}} \rangle$ whose dynamics $\overline{a}_{\mathsf{C}}$ is based on two actions: reset, to set the minimum number of votes needed to report success, and vote to count an individual vote. Formally,

$$\mathsf{reset}\ \langle u, m \rangle\ =\ \iota_1\ \langle m, * \rangle \qquad \text{and} \qquad \mathsf{vote}\ \langle u, * \rangle\ =\ \iota_1\ \langle u-1, u' = 1 \rangle$$

Each incoming vote decreases the concentrator state value. The output of action vote is a boolean flag indicating whether all votes needed to terminate the voting process have been received. A n-voting system is assembled by aggregating n voting pads and connecting their output to the concentrator. A n-codiagonal wire is needed to concentrate the voting pads' outputs. As C is separable, the *hook* combinator can be used for interaction. Thus, we begin with

$$\mathsf{S}_n\ =\ (\boxplus_n \mathsf{VP}\ ;\ulcorner \triangledown_n \urcorner) \boxplus \mathsf{C}$$

which is typed as $\boxplus_n \mathbf{1} + (\mathbb{N} + \mathbf{1}) \longrightarrow \mathbf{1} + (\mathbf{1} + \mathbf{2})$. To apply *hook*, however, S_n has to be wired to exhibit the hooked type in the correct position. Clearly, $\mathsf{S}_n[\mathsf{a}_+, \mathsf{a}_+]$ has the right type: $\boxplus_n(\mathbf{1} + \mathbb{N}) + \mathbf{1} \longrightarrow (\mathbf{1} + \mathbf{1}) + \mathbf{2}$. The voting system is, then, defined as

$$\mathsf{VS}_n\ =\ \big(((\boxplus_n \mathsf{VP}\ ;\ulcorner \triangledown_n \urcorner) \boxplus \mathsf{C})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbf{1}} \quad : \quad \boxplus_n \mathbf{1} + \mathbb{N} \longrightarrow \mathbf{1} + \mathbf{2}$$

Notice that the only actions that are externally available model, respectively, the act of voting (in the voting pad) and system's reset (in the concentrator).

In VS_n each vote is dealt separately. Replacing $\boxplus$ by $\boxast$ as the 'gluing' combinator of the voting pads, allows for the simultaneously counting of arbitrary chunks of votes. Eventually this suits reality better, as several voting pads may be activated at the same time. Two extra modifications are required in the system to cope with this new specification. First, the vote button of the concentrator has to be re-designed to accept, instead of a single stimulus, a natural number encoding the number of incoming votes, *i.e.*,

$$\mathsf{vote}\ \langle u, n \rangle\ = \iota_1\ \langle u-n, u' \leq 0 \rangle$$

Together with the reset button, this defines a new concentrator $\mathsf{NC} : \mathbb{N} + \mathbb{N} \longrightarrow \mathbf{1} + \mathbf{2}$. Second, the codiagonal wiring has to be replaced by a function count_n :

[8] Recall construction $(p\ \rightarrow\ f,\ g)$ is a conditional: if p then f else g.

$\circledast_n \mathbf{1} \longrightarrow \mathbb{N}$ which actually counts the number of inputs received. For $n = 2$, $\mathsf{count}_2 : (\mathbf{1}+\mathbf{1})+\mathbf{1}\times\mathbf{1} \longrightarrow \mathbb{N}$ would simply be $[\underline{1}\cdot!, \underline{2}\cdot!]$. The new system is, then, assembled as in the VS_n case:

$$\mathsf{CVS}_n \;=\; \big(((\circledast_n \mathsf{VP} \,;\, \ulcorner \mathsf{count}_n \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \quad : \quad \circledast_n \mathbf{1} + \mathbb{N} \longrightarrow \mathbf{1}+\mathbf{2}$$

Clearly, CVS_n exhibits a 'richer' behavioural pattern than VS_n, in a very precise sense: if input to CVS_n is restricted to a sum of stimuli, the resulting component becomes bisimilar to VS_n. Let us prove this for $n = 2$ (for $n > 2$ the proof is similar but for some extra wiring manipulation).

Proof.

$$\begin{array}{ll}
& \ulcorner \iota_1 + \mathsf{id} \urcorner \,;\, \mathsf{CVS}_2 \\
\sim & \quad \{\ \mathsf{CVS}_2 \text{ definition and law (14)}\ \} \\
& \big(((\mathsf{VP} \circledast \mathsf{VP} \,;\, \ulcorner \mathsf{count}_n \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}}\,[\iota_1 + \mathsf{id}, \mathsf{id}] \\
\sim & \quad \{\ \text{law (5)}\ \} \\
& \big((((\mathsf{VP} \circledast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_n \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+][\iota_1 + \mathsf{id} + \mathsf{id}, \mathsf{id}]\big)_{\mathbb{N}} \\
\sim & \quad \{\ \mathsf{a}_+ \text{ natural and law (14)}\ \} \\
& \big((\ulcorner \iota_1 + \mathsf{id} \urcorner \,;\, ((\mathsf{VP} \circledast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_2 \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \quad \{\ \text{law (18) and } \mathsf{copy}_X \text{ definition}\ \} \\
& \big(((\ulcorner \iota_1 \urcorner \boxplus \mathsf{copy}_N) \,;\, ((\mathsf{VP} \circledast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_2 \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \quad \{\ \text{law (16)}\ \} \\
& \big(((\ulcorner \iota_1 \urcorner \,;\, ((\mathsf{VP} \circledast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_2 \urcorner)) \boxplus (\mathsf{copy}_N \,;\, \mathsf{NC}))\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \quad \{\ \text{laws (9) and (10)}\} \\
& \big(((\ulcorner \iota_1 \urcorner \,;\, (\mathsf{VP} \circledast \mathsf{VP}) \,;\, \ulcorner \mathsf{count}_2 \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \quad \{\ \text{law (50)}\ \} \\
& \big((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, \ulcorner \iota_1 \urcorner \,;\, \ulcorner \mathsf{count}_2 \urcorner) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \quad \{\ \text{law (10)}\ \} \\
& \big((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, (\ulcorner \iota_1 \urcorner \,;\, \ulcorner \mathsf{count}_2 \urcorner)) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \quad \{\ \mathsf{count}_2 \cdot \iota_1 = \underline{1}\cdot!_{1+1} = \underline{1} \cdot \triangledown\ \} \\
& \big((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, (\ulcorner \triangledown \urcorner \,;\, \ulcorner \underline{1} \urcorner)) \boxplus \mathsf{NC})\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbb{N}} \\
\sim & \quad \{\ \text{law (8)}\ \} \\
& \big((((\mathsf{VP} \boxplus \mathsf{VP}) \,;\, \ulcorner \triangledown \urcorner) \boxplus ((\mathsf{copy}_{\mathbb{N}} \boxplus \ulcorner \underline{1} \urcorner) \,;\, \mathsf{NC}))\,[\mathsf{a}_+, \mathsf{a}_+]\big)_{\mathbf{1}} \\
\sim & \quad \{\ (\mathsf{copy}_{\mathbb{N}} \boxplus \ulcorner \underline{1} \urcorner) \,;\, \mathsf{NC} \sim \mathsf{C}\ \}
\end{array}$$

$$
\begin{array}{ll}
& ((((\mathsf{VP} \boxplus \mathsf{VP})\,;\ulcorner\nabla\urcorner) \boxplus \mathsf{C})\,[\mathsf{a}_+, \mathsf{a}_+])_{\mathbf{1}} \\
\sim & \quad \{\ \mathsf{VS}_2 \text{ definition } \} \\
& \mathsf{VS}_2
\end{array}
$$

$\square$

6 Conclusions and Future Work

This paper introduces a component calculus, intended to help in reasoning (and transforming) component-based designs.

Our notion of a component — stemming from the context of model oriented specification methods — is characterised by the presence of internal state and by an interaction model which reflects the asymmetric nature of input and output. The bicategorical setting adopted is in debt to previous work by R. Walters and his collaborators on models for deterministic input-driven systems [Katis et al., 2000]. However, whereas R. Walters' work deals essentially with deterministic systems, our monadic parametrization allows to focus on the relevant structure of components, factoring out details about the specific behavioural effects that may be produced. The interaction combinators and tensors are also new.

Our approach also enforces a *classification* of component models by the set of laws they satisfy. In fact, although the basic calculus is 'blind' with respect to component's internal structure, being aware of some details of such a structure — for example, some properties of the state space or the specification format of the coalgebra dynamics — enables a finer 'tunning' of the calculus, as illustrated here with the discussion on *separable* components. Another interesting case, currently under investigation, is the class of *restartable* components, *i.e.*, components which may 'die' and be re-activated at a later stage. Further work includes the study of simulation preorders for components as well as of corresponding refinement principles. On the practical side, the prospect of building an *animating tool* for the calculus is currently under consideration. In such a tool, of which an experimental version has been developed in CHARITY [Cockett and Fukushima, 1992], components can be defined and brought to life in order to assess alternative decisions in software architecture [Garlan and Shaw, 1993] design.

References

[Backhouse et al., 1998] Backhouse, R. C., Jansson, P., Jeuring, J., and Meertens, L. (1998). Generic programming: An introduction. In Swierstra, S. D., Henriques, P. R., and Oliveira, J. N., editors, *Third International Summer School on Advanced Functional Programming, Braga*, pages 28–115. Springer Lect. Notes Comp. Sci. (1608).

[Barbosa, 2001] Barbosa, L. S. (2001). *Components as Coalgebras*. PhD thesis, DI, Universidade do Minho.

[Barbosa and Oliveira, 2002] Barbosa, L. S. and Oliveira, J. N. (2002). Coinductive interpreters for process calculi. In *Proc. of FLOPS'02*, pages 183–197, Aizu, Japan. Springer Lect. Notes Comp. Sci. (2441).

[Barbosa and Oliveira, 2003] Barbosa, L. S. and Oliveira, J. N. (2003). State-based components made generic. In Gumm, H. P., editor, *Elect. Notes in Theor. Comp. Sci. (CMCS'03 - Workshop on Coalgebraic Methods in Computer Science)*, volume 82.1, Warsaw.

[Benabou, 1967] Benabou, J. (1967). Introduction to bicategories. *Springer Lect. Notes Maths. (47)*, pages 1–77.

[Bird and Moor, 1997] Bird, R. and Moor, O. (1997). *The Algebra of Programming*. Series in Computer Science. Prentice-Hall International.

[Cockett and Fukushima, 1992] Cockett, R. and Fukushima, T. (1992). About Charity. Yellow Series Report No. 92/480/18, Dep. Computer Science, University of Calgary.

[Garlan and Shaw, 1993] Garlan, D. and Shaw, M. (1993). An introduction to software architecture. In Ambriola, V. and Tortora, G., editors, *Advances in Software Engineering and Knowledge Engineering (volume I)*. World Scientific Publishing Co.

[Goguen et al., 1977] Goguen, J., Thatcher, J., Wagner, E., and Wright, J. (1977). Initial algebra semantics and continuous algebras. *Jour. of the ACM*, 24(1):68–95.

[Jones, 1986] Jones, C. B. (1986). *Systematic Software Development Using* VDM. Series in Computer Science. Prentice-Hall International.

[Katis et al., 2000] Katis, P., Sabadini, N., and Walters, R. F. C. (2000). On the algebra of systems with feedback and boundary. *Rendiconti del Circolo Matematico di Palermo*, II(63):123–156.

[Milner, 1989] Milner, R. (1989). *Communication and Concurrency*. Series in Computer Science. Prentice-Hall International.

[Rutten, 2000] Rutten, J. (2000). Universal coalgebra: A theory of systems. *Theor. Comp. Sci.*, 249(1):3–80. (Revised version of CWI Techn. Rep. CS-R9652, 1996).

[Spivey, 1992] Spivey, J. M. (1992). *The Z Notation: A Reference Manual (2nd ed)*. Series in Computer Science. Prentice-Hall International.

[Szyperski, 1998] Szyperski, C. (1998). *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley.

[Wadler and Weihe, 1999] Wadler, P. and Weihe, K. (1999). Component-based programming under different paradigms. Technical report, Report on the Dagstuhl Seminar 99081.

A Behaviour Monads

A *strong monad* is a monad $\langle \mathsf{B}, \eta, \mu \rangle$ where B is a strong functor and both η and μ are 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. Strength τ_r, followed by τ_l maps $\mathsf{B}I \times \mathsf{B}J$ to $\mathsf{BB}(I \times J)$, which can, then, be flattened to $\mathsf{B}(I \times J)$ via μ. In most cases, however, the *order* of application is relevant for the outcome. The Kleisli composition[9] of the right with the left strength, gives rise to a natural transformation whose component on objects I and J is given by $\delta_r = \tau_{r_{I,J}} \bullet \tau_{l_{\mathsf{B}I,J}}$ Dually, $\delta_l = \tau_{l_{I,J}} \bullet \tau_{r_{I,\mathsf{B}J}}$. Such transformations specify how the monad distributes over product and, therefore, represent a sort of sequential composition of B-computations. Whenever δ_r and δ_l coincide, the monad is said to be *commutative*.

In the approach discussed in this paper, behaviour models for software components are specified by strong monads. Some useful possibilities are:

- *Identity*, $\mathsf{B} = \mathsf{Id}$, yielding *total* and *deterministic* components.
- *Partiality*, *i.e.*, the possibility of deadlock or failure, captured by the maybe monad.
- *Non determinism*, introduced by the (finite) powerset monad, $\mathsf{B} = \mathcal{P}$.
- *Ordered non determinism*, based on the (finite) sequence monad, $\mathsf{B} = \mathsf{Id}^*$.
- Monoidal 'labelling', with $\mathsf{B} = \mathsf{Id} \times M$. Note that, for B to form a monad, parameter M should support a monoidal structure to be used in the definition of η and μ.
- *'Metric' non determinism*, supported on a notion of a *bag* monad based on a structure $\langle M, \oplus, \otimes \rangle$, where both $\oplus$ and $\otimes$ define Abelian monoids over M and the latter distributes over the former. This captures situations in which, among the possible future evolutions of the component, some are more likely (or cheaper, more secure, *etc.*) than others. See [Barbosa and Oliveira, 2003] for particular instantiations.

All of the above situations correspond to known strong monads in Set, which can be composed with each other. The first two and the last one are commutative; the third is not. Commutativity of 'monoidal labelling' depends, of course, on commutativity of the underlying monoid.

The development, in a *point-free* style, of the component calculus discussed in this paper, resorts to a number of laws relating common 'housekeeping' morphisms to cope with *e.g.* product and sum associativity (a and a_+), commutativity (s), right and left units (r and l), right and left distributivity (dr, dl) or exchange (*i.e.*, morphisms $\mathsf{xl} : A \times (B \times C) \longrightarrow B \times (A \times C)$, $\mathsf{xr} : A \times B \times C \longrightarrow A \times C \times B$ and $\mathsf{m} : (A \times B) \times (C \times D) \longrightarrow (A \times C) \times (B \times D)$) with monad unit, multiplication and strength. Such laws are thoroughly dealt with in [Barbosa, 2001] under the designation of *context lemmas*.

[9] Given $f : I \longrightarrow \mathsf{B}J$ and $g : J \longrightarrow \mathsf{B}O$, their Kleisli composition $g \bullet f : I \longrightarrow \mathsf{B}O$ is defined by $g \bullet f = \mu \cdot \mathsf{B}g \cdot f$.

B The Core Calculus

Pipelining

$$\mathsf{copy}_I \,;\, p \;\sim\; p \;\sim\; p \,;\, \mathsf{copy}_O \tag{9}$$

$$(p \,;\, q) \,;\, r \;\sim\; p \,;\, (q \,;\, r) \tag{10}$$

Function lifting

$$\ulcorner f \cdot g \urcorner \sim \ulcorner g \urcorner \,;\, \ulcorner f \urcorner \tag{11}$$

$$\ulcorner \mathsf{id}_I \urcorner \sim \mathsf{copy}_I \tag{12}$$

– any Set isomorphism, split mono and split epi lift to Cp keeping their properties

– $\emptyset$ is *initial* in Bh:

$$\begin{tikzcd} I \arrow[r, "p"] & O \\ \emptyset \arrow[u, "\ulcorner ?_I \urcorner"] \arrow[ur, "\ulcorner ?_O \urcorner"'] & \end{tikzcd} \qquad \ulcorner ?_I \urcorner \,;\, p \sim \ulcorner ?_O \urcorner \tag{13}$$

Wrapping

$$p[f, g] \sim \ulcorner f \urcorner \,;\, p \,;\, \ulcorner g \urcorner \tag{14}$$

$$(p[f, g])[f', g'] \;\sim\; p[f \cdot f', g' \cdot g] \tag{15}$$

Choice

$$(p \boxplus p') \,;\, (q \boxplus q') \;\sim (p \,;\, q) \boxplus (p' \,;\, q') \tag{16}$$

$$\mathsf{copy}_{K \boxplus K'} \;\sim \mathsf{copy}_K \boxplus \mathsf{copy}_{K'} \tag{17}$$

$$\ulcorner f \urcorner \boxplus \ulcorner g \urcorner \;\sim \ulcorner f + g \urcorner \tag{18}$$

$$(p \boxplus q) \boxplus r \;\sim\; (p \boxplus (q \boxplus r))[\mathsf{a}_+, \mathsf{a}_+{}^\circ] \tag{19}$$

$$\mathsf{nil} \boxplus p \;\sim\; p[\mathsf{r}_+, \mathsf{r}_+{}^\circ] \tag{20}$$

$$p \boxplus \mathsf{nil} \;\sim\; p[\mathsf{l}_+, \mathsf{l}_+{}^\circ] \tag{21}$$

$$p \boxplus q \;\sim\; (q \boxplus p)[\mathsf{s}_+, \mathsf{s}_+] \tag{22}$$

Either — $[p, q] = (p \boxplus q) \mathbin{;} \ulcorner \nabla \urcorner$

$$\begin{tikzcd} I \arrow[r, "\ulcorner \iota_1 \urcorner"] \arrow[dr, "p"'] & I \boxplus J \arrow[d, "{[p,q]}"] & J \arrow[l, "\ulcorner \iota_2 \urcorner"'] \arrow[dl, "q"] \\ & O & \end{tikzcd}$$

$$\begin{aligned} \ulcorner \iota_1 \urcorner \mathbin{;} [p, q] &\sim p \\ \ulcorner \iota_2 \urcorner \mathbin{;} [p, q] &\sim q \end{aligned} \quad (23)$$

$$[\ulcorner \iota_1 \urcorner, \ulcorner \iota_2 \urcorner] \sim \mathsf{copy}_{I+J} \quad (24)$$
$$(p \boxplus q) \mathbin{;} [p', q'] \sim [p \mathbin{;} p', q \mathbin{;} q'] \quad (25)$$

$$p \boxplus q \sim [p \mathbin{;} \ulcorner \iota_1 \urcorner, p \mathbin{;} \ulcorner \iota_2 \urcorner] \quad (26)$$
$$\ulcorner \iota_1 \urcorner \mathbin{;} (p \boxplus q) \sim p \mathbin{;} \ulcorner \iota_1 \urcorner \quad (27)$$
$$\ulcorner \iota_2 \urcorner \mathbin{;} (p \boxplus q) \sim q \mathbin{;} \ulcorner \iota_2 \urcorner \quad (28)$$
$$p \mathbin{;} \ulcorner \iota_1 \urcorner \sim \ulcorner \iota_1 \urcorner \mathbin{;} (p \boxplus p) \quad (29)$$

– For any B, however, $\boxplus$ is a weak coproduct in Bh

Parallel

$$\mathsf{copy}_{K \boxtimes K'} \sim \mathsf{copy}_K \boxtimes \mathsf{copy}_{K'} \quad (30)$$
$$(p \boxtimes p') \mathbin{;} (q \boxtimes q') \sim (p \mathbin{;} q) \boxtimes (p' \mathbin{;} q') \quad \text{(if } \mathsf{B} \text{ commutative)} \quad (31)$$

$$\ulcorner f \urcorner \boxtimes \ulcorner g \urcorner \sim \ulcorner f \times g \urcorner \quad (32)$$

$$(p \boxtimes q) \boxtimes r \sim (p \boxtimes (q \boxtimes r))[\mathsf{a}, \mathsf{a}^\circ] \quad (33)$$
$$\mathsf{idle} \boxtimes p \sim p[\mathsf{r}, \mathsf{r}^\circ] \quad (34)$$
$$\mathsf{nil} \boxtimes p \sim \mathsf{nil}[\mathsf{zl}, \mathsf{zl}^\circ] \quad (35)$$
$$p \boxtimes q \sim (q \boxtimes p)[\mathsf{s}, \mathsf{s}] \quad \text{(if } \mathsf{B} \text{ commutative)} \quad (36)$$

Split — $\langle p, q \rangle = \ulcorner \triangle \urcorner \mathbin{;} (p \boxtimes q)$

$$\langle \ulcorner \pi_1 \urcorner, \ulcorner \pi_2 \urcorner \rangle \sim \mathsf{copy}_{O \times R} \quad (37)$$
$$\langle p, q \rangle \mathbin{;} (p' \boxtimes q') \sim \langle p \mathbin{;} p', q \mathbin{;} q' \rangle \quad \text{(if } \mathsf{B} \text{ commutative)} \quad (38)$$

$$p \boxtimes q \sim \langle \ulcorner \pi_1 \urcorner \mathbin{;} p, \ulcorner \pi_2 \urcorner \mathbin{;} q \rangle \quad \text{(if } \mathsf{B} \text{ commutative)} \quad (39)$$

$$(\ulcorner f \urcorner \boxtimes q) \mathbin{;} \ulcorner \pi_2 \urcorner \sim \ulcorner \pi_2 \urcorner \mathbin{;} q \quad (40)$$
$$(p \boxtimes \ulcorner f \urcorner) \mathbin{;} \ulcorner \pi_1 \urcorner \sim \ulcorner \pi_1 \urcorner \mathbin{;} p \quad (41)$$

– In general, *cancellation*

$$\langle p, q\rangle \,;\ulcorner \pi_1 \urcorner \sim p \tag{42}$$

holds only for a monad B which excludes the possibility of *failure* (*e.g.*, the non-empty powerset)

– Diagonal $\triangle$ keeps its naturality when lifted to Cp, for B expressing deterministic behaviour (*e.g.*, the identity or the *maybe* monad), entailing a *fusion* law:

$$r \,; \langle p, q\rangle \sim \langle r \,; p, r \,; q\rangle \tag{43}$$

– $\boxtimes$ is a *product* in Bh, for $\mathsf{B} = \mathsf{Id}$

Concurrent

$$\mathsf{copy}_{K \circledast K'} \sim \mathsf{copy}_K \circledast \mathsf{copy}_{K'} \tag{44}$$

$$(p \circledast p') \,; (q \circledast q') \sim (p \,; q) \circledast (p' \,; q') \quad \text{(if } \mathsf{B} \text{ commutative)} \tag{45}$$

$$(p \circledast q) \circledast r \sim (p \circledast (q \circledast r))[\mathsf{a}_*, \mathsf{a}_*{}^\circ] \tag{46}$$

$$\mathsf{nil} \circledast p \sim p[\mathsf{r}_*, \mathsf{r}_*{}^\circ] \tag{47}$$

$$p \circledast \mathsf{nil} \sim p[\mathsf{l}_*, \mathsf{l}_*{}^\circ] \tag{48}$$

$$p \circledast q \sim (q \circledast p)[\mathsf{s}_*, \mathsf{s}_*] \quad \text{(if } \mathsf{B} \text{ commutative)} \tag{49}$$

$$\ulcorner \iota_1 \urcorner \,; (p \circledast q) \sim (p \boxplus q) \,; \ulcorner \iota_1 \urcorner \tag{50}$$

$$\ulcorner \iota_2 \urcorner \,; (p \circledast q) \sim (p \boxtimes q) \,; \ulcorner \iota_2 \urcorner \tag{51}$$