QoS-aware Component Composition

Luis S. Barbosa
DI-CCTC, Universidade do Minho
4710-057 Braga, Portugal
Email: lsb@di.uminho.pt

Sun Meng
Centrum Wiskunde & Informatica (CWI)
P.O. Box 94079, Amsterdam, The Netherlands
Email: M.Sun@cwi.nl

***Abstract*—Component's QoS constraints cannot be ignored when composing them to build reliable loosely-coupled, distributed systems. Therefore they should be explicitly taken into account in any formal model for component-based development. Such is the purpose of this paper: to extend a calculus of component composition to deal, in an effective way, with QoS constraints. Particular emphasis is put on how the laws that govern composition can be derived, in a calculational, pointfree style, in this new model.**

I. Introduction

The design of loosely-coupled, highly distributed software systems places new requirements on components' composition.

Non-functional properties of software components, such as response time, availability, bandwidth requirement, memory usage, etc., cannot be ignored and become decisive in the selection procedure. Moreover, often adaptation mechanisms have to take them into account, going far behind the simple wrapping of functionality to bridge between published interfaces. The expression *Quality of Service* (QoS) is widely accepted to group together all these concerns [14], [19], [20]. It suggests twin notions of a *level* to be attained and *cost* to be paid, as well as point out to the design of suitable metrics to quantify such properties.

Over the past few decades, several formalisms (e.g., stochastic Petri Nets [13] and interactive Markov Chains [8]) have been proposed to capture different QoS metrics. In programming languages like Java or C#, QoS properties are often specified using meta-attributes. From a static validation perspective, these attributes can be treated like structured comments, which may be used to generate runtime monitors but their semantics is too weak to allow reasoning effectively about QoS properties.

Dealing with QoS aspects in a coherent and systematic way became a main issue in component composition, which cannot be swept under the carpet in any formal account of the problem.

This paper extends a formal calculus for component composition [3], [4], in order to take into account, in an explicit way, QoS information. The calculus is based on a coalgebraic model used to capture components' observable behavior and persistence over transitions. Furthermore, it is parametric on a notion of *behavior*, encoded in a strong monad, which allows to reason in a uniform way about total or partial, non deterministic or stochastic components. It has already been argued by others (e.g., [11], [10], [17]) that coalgebra theory nicely captures the "black-box" characterization of software components, which favors an *observational* semantics: the essence of a component specification lies in the collection of *possible observations* and any two internal configurations should be identified wherever indistinguishable by observation.

To express QoS properties we adopt (a slight generalization) of the notion of Q-algebra proposed in [7]. In brief, a Q-algebra amounts to two semirings over a common carrier, representing some form of *cost* domain, which allows different ways of combining and choosing between quality values.

The resulting calculus provides a compositional approach which offers potential for complex components to be constructed systematically while satisfying QoS constraints. Although most previous laws have to be revisited in this extended model, and most of them turn from equalities (*i.e.*, bisimilarities) to inequalities (*i.e.*, refinements), proofs can still be carried on in the *calculation* style which is the watermark of [3], [4]. This style avoids the explicit construction of, *e.g.*, bisimulations, when proving observational equality, favouring an equational, essentially pointfree reasoning style as popularized, at the *micro*, programming level, under the name of *algebra of programming* [6].

The remaining of this paper is devoted to substantiate this claim. Therefore, the original component calculus is summarised in the following section. Section III introduces its extension and redefines the basic operators to make them *QoS aware*. An illustrative example is discussed in section IV. Section V discusses how the properties of component composition can be established in this new setting. The emphasis is not on providing a complete account of the new calculus, but rather on illustrating how it can be developed.

II. A components' calculus

This section recalls the basic mechanisms for component aggregation along the lines of [3], [4], where further details and proofs can be found.

Software components can be characterized as dynamic systems with a public interface and a private, encapsulated state. The relevance of state information precludes a 'process-like' (purely behavioral) view of components. Components are rather *concrete* coalgebras [17], [1], [11]). For a given value of the state space — referred to as a *seed* in the sequel — a corresponding 'process', or *behavior*, arises by computing its coinductive extension.

In the simplest, deterministic case, the behavior of a component p is captured by the output it produces, which is determined by the supplied input. But reality is often more complicated, for one may have to deal with components whose behavioral pattern is, e.g., partial or even non deterministic. Therefore, to proceed in a generic way, the behavior model is abstracted to a strong monad B. For example, $\mathsf{B} = \mathsf{Id}$ retrieves the simple deterministic behavior, whereas $\mathsf{B} = \mathcal{P}$ or $\mathsf{B} = \mathsf{Id} + \mathbf{1}$ would model non deterministic or partial behavior, respectively[1].

Assume a collection of sets I, O, ..., acting as component interfaces. Then a component taking input in I and producing output in O is specified by a pointed coalgebra

$$\langle u_p \in U_p, \overline{a}_p : U_p \longrightarrow \mathsf{B}(U_p \times O)^I \rangle \tag{1}$$

where u_p is the initial state, often referred to as the *seed*, and the coalgebra dynamics is captured by currying a state-transition function $a_p : U_p \times I \longrightarrow \mathsf{B}\ (U_p \times O)$. This definition means that the computation of an action in a component 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 behavioral effects. Strength, either in its right (τ_r) or left (τ_l) version, cater for context information[2].

Having defined generic components as (pointed) coalgebras, one may wonder how do they get composed and what kind of calculus emerges from this framework. In this framework, interfaces are sets representing the input and output range of a component. Consequently, components are arrows between interfaces and so arrows between components are arrows between arrows. Thus, three notions have to be taken into account: interfaces, components and component morphisms. Formally, this leads to a *bicategorial* setting, but we will avoid such abstraction step in the sequel. For the moment retain that a component morphism $h : \langle u_p, \overline{a}_p \rangle \longrightarrow \langle u_q, \overline{a}_q \rangle$ is just a function connecting the state spaces of p and q and satisfying the following *morphism* and *seed preservation* conditions:

$$\overline{a}_q \cdot h = \mathsf{T}^{\mathsf{B}}\ h \cdot \overline{a}_p \tag{2}$$

$$h\ u_p = u_q \tag{3}$$

Components with compatible interfaces (as in the case $p : I \longrightarrow K$ and $q : K \longrightarrow O$) can be composed sequentially as

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

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 [3]

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

The identity of sequential composition is component $\mathsf{copy}_K : K \longrightarrow K$, where

$$\mathsf{copy}_K = \langle * \in \mathbf{1}, \overline{a}_{\mathsf{copy}_K} \rangle$$

and $a_{\mathsf{copy}_K} = \eta_{\mathbf{1} \times K}$.

Recall (from *e.g.* [17]) that the graph of a morphism is a bisimulation. Therefore, the existence of a seed preserving morphism between two components makes them T^{B}-bisimilar, leading to the following laws, for appropriately typed components p, q and r:

$$\mathsf{copy}_I\,; p \sim p \sim p\,; \mathsf{copy}_O \tag{4}$$

$$(p\,; q)\,; r \sim p\,; (q\,; r) \tag{5}$$

In [3] a collection of component combinators was defined and their properties were studied. In particular it was shown that any function $f : A \longrightarrow B$ can be lifted to a component whose interfaces are given by their domain and codomain types. Formally, a function $f : A \longrightarrow B$ gives rise to component

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

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

$$a_{\ulcorner f \urcorner} = \mathbf{1} \times A \xrightarrow{\mathsf{id} \times f} \mathbf{1} \times B \xrightarrow{\eta_{(\mathbf{1} \times B)}} \mathsf{B}(\mathbf{1} \times B) \tag{6}$$

A *wrapping* mechanism $p[f, g]$ which encodes the pre- and post-composition of a component with functions is defined as a combinator which resembles the renaming connective found in process algebras (*e.g.*, [16]). Let $p : I \longrightarrow O$ be a component and consider functions $f : I' \longrightarrow I$ and $g : O \longrightarrow O'$. Component p wrapped by f and g, denoted by $p[f, g]$ and typed as $I' \longrightarrow O'$, is defined by input pre-composition with f and output post-composition with g. Formally, it maps component p from $\langle u_p, \overline{a}_p \rangle$ into $\langle u_p, \overline{a}_{p[f,g]} \rangle$, where

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

[1] $\mathcal{P}$ is the finite powerset monad. On the other hand, $\mathbf{1}$ represents abstractly a singleton set; therefore type $X + \mathbf{1}$ means *either X or undifined*. The notation used in the sequel is quite standard in mathematics for computer science; reference [6] provides an excellent introduction.

[2] 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 [12]. 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. The Kleisli composition 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.

[3] As one would expect, reasoning about generic components entails a number of laws relating monads with common 'housekeeping' morphisms such as product and sum associativity, (a, a_+), commutativity (s, s_+), left and right units (l, l_+ and r, r_+), left and right distributivity (dl, dr) and isomorphisms $\mathsf{xl} : A \times (B \times C) \longrightarrow B \times (A \times C)$, $\mathsf{xr} : A \times B \times C \longrightarrow A \times C \times B$ and $\mathsf{m} : (A \times B) \times (C \times D) \longrightarrow (A \times C) \times (B \times D)$. Such laws are thoroughly dealt with in [4]. By convention, binary morphisms always associate to the left.

Parallel composition, denoted by $p \boxtimes q$, corresponds to a synchronous product: both components are executed simultaneously when triggered by a pair of legal input values. Note, however, that the behavior effect, captured by monad B, propagates. For example, if B can express component failure and one of the arguments fails, product fails as well. Formally,

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

where

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

and maps every pair of arrows $\langle h_1, h_2 \rangle$ into $h_1 \times h_2$.

There are other generic tensors cater component aggregation. For example, *external choice* $\boxplus$ and *concurrent* $\circledast$ composition. When interacting with $p \boxplus q : I + J \to O + R$, 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, concurrent composition combines choice and parallel, in the sense that p and q can be executed independently or jointly, depending on the input supplied.

Finally, generalized interaction is catered through a sort of "feedback" mechanism on a subset of the inputs. This can be defined by a new combinator, called *hook*, which connects some input to some output wires and, consequently, forces part of the output of a component to be fed back as input. Formally, the *hook* combinator $- \Lsh_Z$ maps each component $p : I + Z \longrightarrow O + Z$ to $p \Lsh_Z : I + Z \longrightarrow O + Z$ given by

$$p \Lsh_Z = \; \langle u_p \in U_p, \overline{a}_{p \Lsh_Z} \rangle$$

where

$$\begin{aligned} a_{p \Lsh_Z} &= U_p \times (I + Z) \xrightarrow{\;a_p\;} \mathsf{B}(U_p \times (O + Z)) \\ &\xrightarrow{\mathsf{B}((\mathsf{id} \times \iota_1 + \mathsf{id} \times \iota_2) \cdot \mathsf{dr})} \mathsf{B}(U_p \times (O + Z) + U_p \times (I + Z)) \\ &\xrightarrow{\mathsf{B}(\eta + a_p)} \mathsf{B}(\mathsf{B}(U_p \times (O + Z)) + \mathsf{B}(U_p \times (O + Z))) \\ &\xrightarrow{\mu \cdot \mathsf{B}\nabla} \mathsf{B}(U_p \times (O + Z)) \end{aligned}$$

III. QoS AS A FIRST CLASS CITIZEN

A. Modelling QoS

QoS is introduced in the component calculus through the notion of a Q-algebra due to [7]. In brief, a *Q-algebra* is an algebraic structure $R = (C, \oplus, \otimes, \oplus\!\!\!\!|, \mathbf{0}, \mathbf{1})$ such that $R_\otimes = (C, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ and $R_{\oplus\!\!\!\!|} = (C, \oplus, \oplus\!\!\!\!|, \mathbf{0}, \mathbf{1})$ are both c-semirings. Intuitively, C is a QoS domain (*e.g.*, a measure of resource usage or availability) whereas $\oplus$ represents a *choice* between QoS values and $\otimes$ and $\oplus\!\!\!\!|$, respectively, compose QoS values sequentially or concurrently.

This definition entails distribution of both $\otimes$ and $\oplus\!\!\!\!|$ over $\oplus$, and an absorption law again for both combinators, *i.e.*,

$$(a \oplus b) \circ c \;=\; (a \circ c) \oplus (b \circ c) \qquad (7)$$

$$\mathbf{0} \circ a \;=\; \mathbf{0} \qquad (8)$$

where $\circ = \otimes, \oplus\!\!\!\!|$. Clearly, $\oplus$ defines a partial order

$$a \leq b \;\Leftrightarrow\; (a \oplus b) = b \qquad (9)$$

meaning a is *worse* than b, *i.e.*, has a higher cost. Alternatively, $\oplus$ could be derived from a semi-lattice $(C, \leq)$ as the corresponding least upper bound.

With respect to [7], we additionally require that both $\otimes$ and $\oplus\!\!\!\!|$ be commutative, which makes easy formal manipulation in proofs. More fundamentally, we also require that the common identity of these cost combinators acts as an absorving element wrt $\oplus$, *i.e.*

$$\mathbf{1} \oplus a \;=\; \mathbf{1} \qquad (10)$$

Law (10) is required to establish essential properties relating the QoS levels of individual components with their composition, in particular that $a \circ b \leq a$, for $\circ = \otimes, \oplus\!\!\!\!|$. Actually,

$$\begin{aligned} & a \circ b \leq a \\ \Leftrightarrow \quad & \quad \{ \text{(9)} \} \\ & (a \circ b) \oplus a = a \\ \Leftrightarrow \quad & \quad \{ \text{identity for } \circ \} \\ & (a \circ b) \oplus (a \circ \mathbf{1}) = a \\ \Leftrightarrow \quad & \quad \{ \text{(7)} \} \\ & a \circ (b \oplus \mathbf{1}) = a \\ \Leftrightarrow \quad & \quad \{ \text{(10)} \} \\ & a \circ \mathbf{1} = a \\ \Leftrightarrow \quad & \quad \{ \text{identity for } \circ \} \\ & a = a \end{aligned}$$

Section IV presents concrete examples of this sort of structure. Further examples are discussed, although in a different context, in [2].

B. QoS-aware components

QoS information is included in the component model as an additional attribute: its execution generates a QoS value which is observable (*i.e.*, measurable). Formally, definition (1) changes to

$$\langle u_0 \in U_p, \overline{\alpha}_p : U_p \to \mathsf{B}(U_p \times C \times O)^I \rangle \qquad (11)$$

where C is the domain of some Q-algebra $R = (C, \oplus, \otimes, \oplus\!\!\!\!|, \mathbf{0}, \mathbf{1})$.

The definition of all component combinators change accordingly to take into account the need for equally combining the observed QoS levels of their parameters. The dynamics of sequential composition becomes

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

Note the use of $\otimes$ to sequentially compose QoS levels. The same occurs in the redefinition of the *hook* combinator $-\urcorner_Z$, which is essentially a generalization of sequential composition.

The redefinition of parallel composition, on its turn, resorts to $\oplus$:

$$\begin{aligned} \alpha_{p \boxtimes q} &= U_p \times U_q \times (I \times J) \xrightarrow{\mathsf{m}} (U_p \times I) \times (U_q \times J) \\ &\xrightarrow{\alpha_p \times \alpha_q} \mathsf{B}(U_p \times C \times O) \times \mathsf{B}(U_q \times C \times K) \\ &\xrightarrow{\delta_l} \mathsf{B}((U_p \times C \times O) \times (U_q \times C \times K)) \\ &\xrightarrow{\mathsf{Bm}} \mathsf{B}((U_p \times C) \times (U_q \times C) \times (O \times K)) \\ &\xrightarrow{\mathsf{B}(\mathsf{m} \times \mathsf{id})} \mathsf{B}((U_p \times U_q) \times (C \times C) \times (O \times K)) \\ &\xrightarrow{\mathsf{B}(\mathsf{id} \times \oplus \times \mathsf{id})} \mathsf{B}(U_p \times U_q \times C \times (O \times K)) \end{aligned}$$

Although not discussed in detail in this paper, component *choice*, $\boxplus$, requires the introduction of a new operator. Actually, the global QoS level of $p \boxplus q$ is computed as $c_1 \odot c_2$, where $\odot$ is the glb of order $\leq$. Actually, the cost order has the structure of a lattice whose lub gives $\oplus$ and glb gives $\odot$.

C. And equalities become refinements ...

In the presence of QoS constraints most bisimulation equations become *refinements*, in the sense the word has in coalgebra theory [15]. Note that a bisimulation over $\mathsf{T}X = \mathsf{B}(X \times C \times O)^I$, which acts as the notion of equality in the calculus, entails, as usual, syntactic equality at the interface level, and thus equal costs. To compare component-based designs with respect to QoS measures entails the need for a notion of *refinement*: $p \trianglelefteq q$ (read component q refines p) if they exhibit the same behavior but the QoS level of q is higher (*i.e.*, "costs less") than that of p. Formally, this is recorded by the existence of what was called in [15] a *forward* morphism with respect to an order $\sqsubseteq$ which captures QoS increase. We will briefly review its definition and precisely characterize the refinement order relevant for reasoning about QoS-aware components.

Recall first that a T-coalgebra morphism $h : q \longrightarrow p$ is a function from the state space of q to that of p such that

$$\mathsf{T}h \cdot q = p \cdot h \tag{12}$$

It is well known that the existence of such a morphism entails bisimulation. A weak form of morphism where (12) is rephrased in terms of a refinement preorder over the image of functor T was proposed in [15]:

$$\mathsf{T}h \cdot q \ \dot{\sqsubseteq}\ p \cdot h \tag{13}$$

where[4] it was proved such a morphism still preserves transitions, therefore entailing a form of simulation, and that for a given functor T coalgebras and forward morphisms still form a category. Preorder $\sqsubseteq$ cannot be arbitrary, but compatible with the *structural membership* $\in_\mathsf{T}$ defined for regular functors in [9]. This means that $(\in_\mathsf{T} \cdot \sqsubseteq) \subseteq \in_\mathsf{T}$, or, equivalently, $\sqsubseteq \subseteq (\in_\mathsf{T} \setminus \in_\mathsf{T})$, *i.e.*, the relational division of structural membership by itself. A typical refinement preorder, called *structural inclusion*, is given by

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

Note that $\subseteq_\mathsf{T}$ captures the classical notion of *non determinism reduction*. The refinement preorder relevant to capture increase in component's QoS level is obtained from *structural inclusion* by replacing the second clause by

$$x \subseteq_K y \quad \text{iff} \quad \begin{cases} K = C & \Rightarrow x \sqsubseteq y \\ K \neq C & \Rightarrow x =_K y \end{cases}$$

Clearly,

$$\subseteq_\mathsf{T} \ \subseteq\ \sqsubseteq_\mathsf{T}\ \subseteq\ (\in_\mathsf{T} \setminus \in_\mathsf{T}) \tag{16}$$

Section V discusses in detail the rendering of what was in the original component calculus an equivalence law, as a refinement inequation wrt $\sqsubseteq_\mathsf{T}$.

IV. An example

Both the component calculus and QoS modelling are illustrated in this section through a well-known example: the Alternating Bit Protocol (ABP) [18]. It is a simple communication protocol that provides an error-free communication over a medium that might lose messages. The description consists of four components: a sender and a receiver which are connected

[4]We denote by $\dot{\sqsubseteq}$ the pointwise lifting of $\sqsubseteq$ to the functional level,

$$f\ \dot{\sqsubseteq}\ g \ \Leftrightarrow\ \forall_x.\ f\ x \sqsubseteq g\ x \tag{14}$$

which can also be formulated in the following pointfree way,

$$f\ \dot{\sqsubseteq}\ g \ \Leftrightarrow\ (f \subseteq \sqsubseteq \cdot g) \tag{15}$$

via two media. The structure of the communication system are shown in Figure 1.

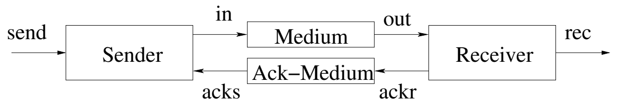

Fig. 1. The Structure of ABP

A retransmission mechanism is used to overcome the unreliability of the medium. When the sender observes a delay between issuing a message to the medium (via the in channel) and receiving an acknowledgement (via the $acks$ channel) that it deems too long, it initiates a retransmission (we assume that the retransmission happens when the delay is t time units). In this model we assume that sometimes messages are lost in the media and some time delays occur in the media as well. So the QoS properties we are interested here are reliability and time. These two quality dimensions are represented by the corresponding Q-algebras

$$\begin{aligned} Q_1 &= ([0,1], max, \times, \times, 0, 1) \\ Q_2 &= (\mathbb{R}_+ \cup \{\infty\}, min, +, max, \infty, 0) \end{aligned}$$

Furthermore, they can be composed to yield a new Q-algebra for the two QoS dimensions. Elements of this composed structure are pairs of elements of Q_1 and Q_2. Formally

$$\begin{aligned} Q &= ([0,1] \times (\mathbb{R}_+ \cup \{\infty\}), \\ &\quad \langle max, min\rangle, \langle \times, +\rangle, \langle \times, max\rangle, \langle 0, \infty\rangle, \langle 1, 0\rangle) \end{aligned}$$

Let $C = [0,1] \times (\mathbb{R}_+ \cup \{\infty\})$ be the domain of Q, and $T = \mathbb{R}_+ \cup \{\infty\}$ be the set of time values, we define the four components as follows [5]:

$$\begin{aligned} &\mathsf{Sender} : U \longrightarrow (U \times C \times (M \times \mathbf{2}))^{(M + M \times \mathbf{2} + T)} \\ &\mathsf{Sender} \;=\; \langle * \in U, \overline{a}_{\mathsf{Sender}}\rangle \end{aligned}$$

where $U = (M \times \mathbf{2}) \cup \mathbf{1}$, and

$$a_{\mathsf{Sender}}\ \langle u, i\rangle = \begin{cases} \langle\langle m,0\rangle, c_{\mathsf{Sender}}, \langle m,0\rangle\rangle & \text{if } u = * \wedge i = m \\ \langle\langle m,0\rangle, c_{\mathsf{Sender}}, \langle m,0\rangle\rangle & \text{if } u = \langle m,0\rangle \wedge i = t \\ \langle\langle m',1\rangle, c_{\mathsf{Sender}}, \langle m',1\rangle\rangle & \text{if } u = \langle m,0\rangle \wedge i = \langle m',0\rangle \\ \langle\langle m',1\rangle, c_{\mathsf{Sender}}, \langle m',1\rangle\rangle & \text{if } u = \langle m',1\rangle \wedge i = t \\ \langle\langle m,0\rangle, c_{\mathsf{Sender}}, \langle m,0\rangle\rangle & \text{if } u = \langle m',1\rangle \wedge i = \langle m,1\rangle \end{cases}$$

$$\begin{aligned} &\mathsf{Medium} : \mathbf{1} \times (M \times \mathbf{2}) \longrightarrow \mathbf{1} \times C \times (M \times \mathbf{2}) \\ &\mathsf{Medium} \;=\; \langle * \in \mathbf{1}, \overline{a}_{\mathsf{Medium}}\rangle \end{aligned}$$

where

$$a_{\mathsf{Medium}}\ \langle *, \langle m,k\rangle\rangle \;=\; \langle *, c_{\mathsf{Medium}}, \langle m,k\rangle\rangle$$

[5]Since the behavior of these components is functional and deterministic, B is obviously the identity monad Id, which is omitted to simplify writing.

$$\begin{aligned} &\mathsf{AckMedium} : \mathbf{1} \times \mathbf{2} \longrightarrow \mathbf{1} \times C \times \mathbf{2} \\ &\mathsf{AckMedium} \;=\; \langle * \in \mathbf{1}, \overline{a}_{\mathsf{Ack}}\rangle \end{aligned}$$

where

$$a_{\mathsf{Ack}}\ \langle *, k\rangle \;=\; \langle *, c_{\mathsf{Ack}}, k\rangle$$

Component Receiver is defined as

$$\mathsf{Receiver} \;=\; \triangle\ ;\ \mathsf{outrec} \boxtimes \mathsf{outackr}$$

where

$$\begin{aligned} &\mathsf{outrec} : \mathbf{2} \times M \times \mathbf{2} \longrightarrow \mathbf{2} \times (C \times M + C) \\ &\mathsf{outrec} \;=\; \langle 0 \in \mathbf{2}, \overline{a}_{\mathsf{outrec}}\rangle \end{aligned}$$

$$a_{\mathsf{outrec}}\ \langle u, i\rangle = \begin{cases} \langle 1, c_{\mathsf{outrec}}, m\rangle & \text{if } u = 0 \wedge i = \langle m,0\rangle \\ \langle 0, c_{\mathsf{outrec}}, m\rangle & \text{if } u = 1 \wedge i = \langle m,1\rangle \\ \langle 0, c_{\mathsf{outrec}}\rangle & \text{if } u = 1 \wedge i = \langle m,0\rangle \\ \langle 1, c_{\mathsf{outrec}}\rangle & \text{if } u = 0 \wedge i = \langle m,1\rangle \end{cases}$$

$$\begin{aligned} &\mathsf{outackr} : \mathbf{2} \times M \times \mathbf{2} \longrightarrow \mathbf{2} \times (C \times \mathbf{2}) \\ &\mathsf{outackr} \;=\; \langle 0 \in \mathbf{2}, \overline{a}_{\mathsf{outackr}}\rangle \end{aligned}$$

$$a_{\mathsf{outackr}}\ \langle u, i\rangle = \begin{cases} \langle 1, c_{\mathsf{outackr}}, 0\rangle & \text{if } u = 0 \wedge i = \langle m,0\rangle \\ \langle 0, c_{\mathsf{outackr}}, 1\rangle & \text{if } u = 1 \wedge i = \langle m,1\rangle \\ \langle 1, c_{\mathsf{outackr}}, 0\rangle & \text{if } u = 1 \wedge i = \langle m,0\rangle \\ \langle 0, c_{\mathsf{outackr}}, 1\rangle & \text{if } u = 0 \wedge i = \langle m,1\rangle \end{cases}$$

The whole system is written as

$$\mathsf{ABP} \;=\; (\mathsf{Sender}\ ;\ \mathsf{Medium}\ ;\ \mathsf{Receiver}\ ;\ \mathsf{AckMedium}) \urcorner_{\mathbf{2}}$$

The *quality improving* refinement order discussed in the previous section can be used here to establish a relationship between different implementations of the APB protocol. Monotonicity of $\sqsubseteq_{\mathsf{T}}$, which one gets for free once proved it is a coalgebraic refinement relation (by (16)), ensures that replacing, say, the Sender component by a faster one, or the component representing the communication medium by a reliable one, *i.e.*, $\mathsf{Sender}' \trianglelefteq \mathsf{Sender}$ and $\mathsf{Medium}' \trianglelefteq \mathsf{Medium}$, one gets

$$(\mathsf{Sender}'\ ;\ \mathsf{Medium}'\ ;\ \mathsf{Receiver}\ ;\ \mathsf{AckMedium}) \urcorner_{\mathbf{2}} \;\trianglelefteq\; \mathsf{ABP}$$

Next section puts this problem in a broader context, illustrating how an equational law on component composition gives rise to a refinement result in a QoS-aware model.

V. Revisiting component laws

For space limitations, we limit ourselves to mention a few, hopefully illustrative, examples. As mentioned in section II, the basic mechanism provided in the calculus to adapt component interfaces is *wrapping* and amounts to the pre and post composition of a component with suitable functions which modify the input and output universes without interfering in the component dynamics. The QoS level of $p[f,g]$ is, of course, that of p.

The calculus also allows to regard functions as particular instances of components, whose interfaces are given by their domain and codomain types. As also discussed above, a function $f : A \longrightarrow B$ is represented by component $\ulcorner f \urcorner$ Assigning a cost (or generically a QoS measure) $c \in C$ to the execution of f corresponds to change (6) to

$$a_{\ulcorner f \urcorner} = \mathbf{1} \times A \xrightarrow{\langle \mathsf{id}, \underline{c} \rangle \times f} \mathbf{1} \times C \times B \xrightarrow{\eta_{(\mathbf{1} \times B)}} \mathsf{B}(\mathbf{1} \times C \times B)$$

What is the effect of QoS annotations in the calculus? Clearly, properties like

$$p[f,g] \sim \ulcorner f \urcorner \,;\, p \,;\, \ulcorner g \urcorner \tag{17}$$

$$(p[f,g])[f',g'] \sim p[f \cdot f', g' \cdot g] \tag{18}$$

only hold if functions are lifted at no cost (*i.e.*, with perfect QoS). In this new setting, equation (17) is reduced to a refinement inequality

$$\ulcorner f \urcorner \,;\, p \,;\, \ulcorner g \urcorner \;\trianglelefteq\; p[f,g] \tag{19}$$

Parallel composition of components provides another interesting source of examples of previous equational laws lifting to refinement inequations. A basic question, which moreover is independent of QoS introduction, is whether $\boxtimes$ lifts to a universal product construction at the behavioral level, *i.e.*, whether it behaves like Cartesian product when modelling types by simple sets. Therefore we start by definning the *split* of two components as

$$\langle p, q \rangle \;=\; \ulcorner \triangle \urcorner \,;\, (p \boxtimes q) \quad \text{where} \quad \triangle = \langle id, id \rangle$$

exactly as the split of two sets (formally, the *universal* function to the Cartesian product). This definition, however, does not guarantee, in general, the commutativity of

$$\begin{tikzcd} & I \arrow[dl, "p"'] \arrow[d, "\langle p,q \rangle"] \arrow[dr, "q"] & \\ O & O \boxtimes R \arrow[l, "\ulcorner \pi_1 \urcorner"] \arrow[r, "\ulcorner \pi_2 \urcorner"'] & R \end{tikzcd} \tag{20}$$

Qualifier *in general* above means *for any monad* B; obviously (20) holds for *deterministic* components (*i.e.*, with $\mathsf{B} = \mathsf{Id}$), as in the example of section IV. One may prove, however, that, for a broad range of (commutative) monads a *cancellation* law

$$p \;\sim\; \langle p, q \rangle \,;\, \ulcorner \pi_1 \urcorner \tag{21}$$

holds. In the extended QoS-aware calculus, equation (21) is only a refinement

$$p \;\trianglelefteq\; \langle p, q \rangle \,;\, \ulcorner \pi_1 \urcorner \tag{22}$$

because the right-hand side entails the execution of both p and q, thus $\otimes$-composing the respective costs.

VI. Conclusions

We have shown how, with a slight generalization of the notion of Q-algebra proposed in [7], a component calculus can be extended, in a *systematic* way, to deal with QoS measures. This shows that reasoning formally about QoS-aware components is feasible; the resulting calculus being a smooth extension of the original one.

This leads to a new dimension on refinement, which one may call *quality improving refinement*, as a way to guarantee not only the functional simulation relation induced by the behavior of components, as in [15], [5], but also a higher (or at least equal) service quality.

The extended component calculus is doubled parametric in both dimensions: *behavior* (through a strong monad B) and *QoS* (through a Q-algebra). Therefore, it offers a suitable setting for reasoning, at a high level of abstraction, about component composition and, in general, coordination problems. Whether and how it scales up to composition of mobile components and their dynamic reconfiguration, is the topic of our current research.

References

[1] J. Adamek. An introduction to coalgebra. *Theory and Applications of Categories*, 14(8):157–199, 2005.

[2] F. Arbab, T. Chothia, M. Sun, and Y.-J. Moon. Component Connectors with QoS Guarantees. In A. L. Murphy and J. Vitek, editor, *Proceedings of 9th International Conference on Coordination Models and Languages, Coordination'07*, volume 4467 of *LNCS*, pages 286–304. Springer, 2007.

[3] L. S. Barbosa. Towards a Calculus of State-based Software Components. *Journal of Universal Computer Science*, 9(8):891–909, August 2003.

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

[5] L. S. Barbosa and J. N. Oliveira. Transposing partial components: an exercise on coalgebraic refinement. *Theor. Comp. Sci.*, 365(1-2):2–22, 2006.

[6] R. Bird and O. de Moor. *Algebra of Programming*. Prentice Hall, 1997.

[7] T. Chothia and J. Kleijn. Q-automata: Modelling the resource usage of concurrent components. *Electronic Notes in Theoretical Computer Science*, 175(2):153–167, 2007.

[8] H. Hermanns. *Interactive Markov Chains And the Quest for Quantified Quality*. Springer, 2002.

[9] P. F. Hoogendijk. *A generic theory of datatypes*. PhD thesis, Department of Computing Science, Eindhoven University of Technology, 1996.

[10] B. Jacobs. Objects and classes, co-algebraically. In B. Freitag, C. L. C.B. Jones, and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer, 1996.

[11] B. Jacobs. Exercises in coalgebraic specification. In R. Backhouse, R. Crole, and J. Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*, volume 2297 of *LNCS*, pages 237–280. Springer, 2002.

[12] A. Kock. Strong functors and monoidal monads. *Archiv für Mathematik*, 23:113–120, 1972.

[13] M. A. Marsan, G. Conte, and G. Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[14] D. A. Menascé. Composing Web Services: A QoS View. *IEEE Internet Computing*, 8(6):88–90, 2004.

[15] S. Meng and L. S. Barbosa. Components as Coalgebras: the Refinement Dimension. *Theoretical Computer Science*, 351(2):276–294, 2006.

[16] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[17] J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

[18] K. Tarnay. *Protocol Specification and Testing*. Plenum Press, 1991.

[19] T. Yu and K.-J. Lin. Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In B. Benatallah, F. Casati, and P. Traverso, editors, *ICSOC 2005*, volume 3826 of *LNCS*, pages 130–143. Springer, 2005.

[20] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.