Revisiting context-aware component interconnection

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

Marco A. C. Barbosa
Universidade de Cruz Alta
Brazil
Email: marco.cb@gmail.com

César J. Rodrigues
DI-CCTC, Universidade do Minho
4710-057 Braga, Portugal
Email: cjr@di.uminho.pt

***Abstract*—Software connectors are external coordination devices which ensure the flow of data and enforce synchronization constraints within a component's network. The specification of software connectors through which context dependent behaviour is correctly propagated remains an open, non trivial issue in their semantics. This paper, building on previous work by the authors, revisits this problem and introduces a model in which context awareness is suitably handled.**

I. Introduction

The design of loosely-coupled, highly distributed software systems places new requirements on components' composition. A main issue, actually not new but increasingly relevant, concerns decoupling of specific *loci* of computation from the protocols that govern their interaction to achieve common tasks.

Typically, such *loci* of computation, referred to as *components* in the sequel, are regarded as black-boxes, and characterized by a set of ports through which data values are sent or received. Ports have a polarity (either *input* or *output*) and, in most cases, a type to classify the admissible values. The underlying coordination discipline is *exogenous* in the precise sense that each component is unaware of which other components it is connected to and under which protocols. Component coordination becomes the purpose of special devices intended to regulate the flow of data and enforce synchronization constraints: such is the role of *software connectors* [11], [17].

Reo [1], [2] is a well-known example of an exogenous coordination language built around a notion of connectors as a first class citizen, but by no means the only one. Orc [12], and, in general, high-level composition abstractions proposed for what is known as *world-programming* languages [10], often inspired by the process algebra legacy, also fall into this category.

An issue which is clearly non trivial in the semantics software connectors is known as *context awareness*. The expression refers to the ability to capture behaviours which include non-monotonic choices determined by the way their environments, consisting of pending activity on their ports, changes. The big challenge is to ensure the correct propagation of such context-dependent behaviour.

This notion, even if in general is difficult to formalise, is regarded as fundamental in exogenous coordination research and as such is considered in the informal semantics of Reo as presented in its foundational papers [1], [2]. Even though, and in the context of Reo the problem was not solved in the two main formal semantics proposed for the language based on timed data streams [3] and constraint automata [4]. A partial solution was put forward in the so-called connector-coulouring semantics [8], which is highly operational and admits degenerated behaviour in a number of cases. Formal semantics for Reo context-aware connectors were proposed quite recently in [7] and [9], independently of each other.

In [6] the authors proposed a model in which a connector's model is decomposed into two distinct specifications: a typed binary relation to capture the flow of data, and a behavioural pattern to describe its interface. The latter amounts to a specification of which, when and under what conditions connector's ports become activated (*i.e.*, ready to deliver or consume a datum). The specification of connectors' behavioural patterns resorts to a process calculus parametric on the communication discipline fully developed in [15]. In [6] the same language is also used to describe component interfaces, i.e. the *business protocols* they implement, enabling local and global, compositional reasoning over applications.

Connector construction in this approach is compositional: new connectors are built out of old, through six specific connectives: parallel and concurrent *aggregation*, *interleaving*, left and right port *join* and *hook*, the latter corresponding to a sort of feedback mechanism.

The propagation of context-dependent behaviour through a composite connector was only partially handled in the model proposed in [6]. This paper is, therefore, an attempt to further develop this approach to express context dependency, retaining the calculational, essentially equational reasoning style which is the main distinguished characteristic of [6] with respect to automata-based approaches used in other semantical models for Reo.

The new model is introduced in the next two sections. Section IV discusses its suitability to correctly deal with context awareness. The model at work is illustrated in section V. Finally, section VI concludes and highlights some topics for further research.

II. Modelling connectors

A. Connectors

In loosely-coupled systems components cooperate through specific *connectors* which abstract the idea of intermediate *glue code* to handle interaction. This section tackles the question *what connectors are and how do they compose*,

proposing a model which improves on the one underlying [6] to cater, in an effective way, context-awareness.

As usual, connectors have ports, i.e., *interface points* through which messages flow. Each port has an *interaction polarity*, either *input* or *output*, but, in general, connectors are blind with respect to the data values flowing through them. Formally, let $\mathbb{C}$ be a connector with m input and n output ports. Assume $\mathbb{M}$ is a generic type of messages and $\mathbb{P}$ a set of (unique) *port identifiers*. In a number of cases it is necessary to consider a default value in $\mathbb{D}$ to represent *absence* of messages, for example to describe a transition in a connector's state in which a particular port is not involved. Therefore, the type of data $\mathbb{D}$ flowing through connectors is defined as

$$\mathbb{D} \triangleq \mathbb{M} + \mathbf{1} \tag{1}$$

where $\mathbf{1}$ is the singleton set whose unique element is represented, by convention, as $\perp$.

Elementary connectors are stateless, but to introduce asynchrony, *e.g.*, through a buffered channel, internal states might be considered. Let $\mathbb{U}$ stand for a generic type of state spaces, typically given as a functorial expression in $\mathbb{D}$. For example $\mathbb{U}$ may be defined as a sequence of data ($\mathbb{U} = \mathbb{D}^*$) or, as in the specification of a one-fifo buffer below, simply as $\mathbb{U} = \mathbb{D}$. Default value $\perp$ stands now for absence of stored information in the connectors memory. Formally, the behaviour of a connector is defined as follows

Definition 1: The specification of a connector $\mathbb{C}$ is given by a relation

$$\mathsf{data}.[\![\mathbb{C}]\!] : \mathbb{D}^n \times \mathbb{U} \longleftarrow \mathbb{D}^m \times \mathbb{U} \tag{2}$$

which records the flow of data, and a process expression

$$\mathsf{port}.[\![\mathbb{C}]\!] \in Bhv \tag{3}$$

which gives the behavioural pattern for port activation.

Bhv is the process language generated according to the following grammar:

$$P ::= \mathbf{0} \mid a \cdot P \mid P + P \mid P \boxtimes P \mid P \boxplus P \mid P \boxdot P \mid \mathsf{fix}\ (X = P)$$

where a is an element of $\mathbb{A}$. In a first approximation the set $\mathbb{A}$, of actions, is defined as $\mathbb{A} = \mathcal{P}(\mathbb{P} \cup \{\tau\})$, *i.e.*, as sets of connectors' ends plus a special symbol, τ, to represent any unobservable action. The introduction of τ is technically entailed by the semantics of the *hook* combinator, as explained below. Regarded as an action, a port identifier a asserts the activation of the corresponding port, i.e., the fact that a datum crosses its boundaries. Note that choosing $\mathbb{A}$ as a *set* of port identifiers allows for the synchronous activation of several ports in a single computational step.

B. Behavioural patterns

The semantics of Bhv expressions used to capture connectors behaviour or interaction protocol, is fairly standard, but for the parametrization of all forms of parallel composition (*i.e.*, $\boxtimes$ and $\boxdot$) by an interaction discipline as discussed below. The remaining combinators have the usual meaning as in, e.g. CCS [14].

The idea of using behaviour-annotated interfaces for connectors is not new, but common to most modern architectural description languages. Process algebra provides an expressive setting for representing behavioural patterns and establish or verify their properties in a compositional way. Each process algebra introduces a number of combinators for processes (or *behaviours*) and an interaction discipline, for example synchronisation of complementary labels in CCS [14]. In the context of component coordination, however, sticking to a fixed interaction discipline is a severe limitation: as shown in [6], different such disciplines can be used, at the same time, to capture different aspects of component coordination.

To meet this goal, which entails the need for a *generic* way to *design* process algebras, we built on top of our previous work [5], [15]. There processes are identified with inhabitants of a final coalgebra and their combinators defined by *coinductive extension* [16] of 'one-step' behaviour generator functions [1].

A typical example is the *interleaving* $\boxplus : \nu \longleftarrow \nu \times \nu$ combinator, an interaction-free form of parallel composition. Its definition, in Fig. 1, captures the intuition that global observations correspond to all possible interleavings of local observations. Morphisms τ_r and τ_l stand for, respectively, the right and left *strength* [13] associated to functor $\mathcal{P}(\mathbb{A} \times \mathsf{Id})$, whose effect amounts to a distributive law.

The *synchronous product* models the simultaneous execution of two processes, which, in each step, interact through the actions they realize. Let us, for the moment, represent such interaction by a function $\theta : \mathbb{A} \times \mathbb{A} \longleftarrow \mathbb{A}$. The formal definition is given, again, in Fig. 1, where function sel filters out all synchronisation failures and δ_r is given by

$$\delta_r\ \langle c_1, c_2 \rangle = \{\langle a'\ \theta\ a, \langle p, p' \rangle \rangle |\ \langle a, p \rangle \in c_1 \wedge \langle a', p' \rangle \in c_2\}$$

Defined over $\mathbb{A}$, θ captures the envisaged interaction discipline, endowing $\mathbb{A}$ with the structure of an Abelian positive monoid $\langle \mathbb{A}; \theta, 1 \rangle$ with a zero element 0. The intuition is that θ determines the interaction discipline whereas 0 represents the absence of interaction: for all $a \in \mathbb{A}$, $a\theta 0 = 0$.

Synchronous product depends in a crucial way on the interaction structure adopted. For example its commutativity depends only on the commutativity of the underlying θ. Such is also the case of *parallel composition* which combines the effects of both $\boxplus$ and $\boxtimes$. Note, however, that such a combination is performed at the *genes* level, as shown again in Fig. 1.

C. Examples

We consider now a few examples of (REO-based) software connectors.

Synchronous channel. The *synchronous channel* has two ports of opposite polarity. This connector forces input and

[1] Technically, this amounts to the systematic use of the universal property which characterizes coinductive extensions. Recall that, for a functor T and an arbitrary coalgebra $\langle U, p : \mathsf{T}\,U \longleftarrow U \rangle$, its *coinductive extension*, represented by $[\![p]\!]$, is the *unique* morphism from p to the final coalgebra $\omega_{\mathsf{T}} : \mathsf{T}\ \nu_{\mathsf{T}} \longleftarrow \nu_{\mathsf{T}}$.

$$\boxplus = [\![(\alpha_{\boxplus})]\!] \text{ where } \alpha_{\boxplus} = \nu \times \nu \xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \xrightarrow{(\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)} (\mathcal{P}(\mathbb{A} \times \nu) \times \nu) \times (\nu \times \mathcal{P}(\mathbb{A} \times \nu))$$
$$\xrightarrow{\tau_r \times \tau_l} \mathcal{P}(\mathbb{A} \times (\nu \times \nu)) \times \mathcal{P}(\mathbb{A} \times (\nu \times \nu)) \xrightarrow{\cup} \mathcal{P}(\mathbb{A} \times (\nu \times \nu))$$
$$\boxtimes = [\![(\alpha_{\boxtimes})]\!] \text{ where } \alpha_{\boxtimes} = \nu \times \nu \xrightarrow{(\omega \times \omega)} \mathcal{P}(\mathbb{A} \times \nu) \times \mathcal{P}(\mathbb{A} \times \nu) \xrightarrow{\mathsf{sel} \cdot \delta_r} \mathcal{P}(\mathbb{A} \times (\nu \times \nu))$$
$$\boxdot = [\![(\alpha_{\boxdot})]\!] \text{ where } \alpha_{\boxdot} = \cup \cdot (\alpha_{\boxplus} \times \alpha_{\boxtimes}) \cdot \Delta$$

Fig. 1. Definition of $\boxplus$, $\boxtimes$ and $\boxdot$ by coinductive extension.

output to become mutually blocking, in the sense that any of them must wait for the other to be completed.

$$\mathsf{data}.[\![\ a \longrightarrow b\ ]\!] = \mathsf{Id}_{\mathbb{D} \times \mathbb{U}}$$
$$\mathsf{port}.[\![\ a \longrightarrow b\ ]\!] = \mathsf{fix}\,(X = ab \cdot X)$$

Here, as well as in the next three cases, state information is irrelevant. Therefore, $\mathbb{U} = \mathbf{1}$. Its relational, data semantics is simply the identity relation on data domain $\mathbb{D}$ and its behaviour is captured by the simultaneous activation of its two ports.

Drain. A drain has two input, but no output, ports. Therefore, it looses any data item crossing its boundaries. Formally,

$$\mathsf{data}.[\![\ a \longmapsto\!\dashv b\ ]\!] = (\mathbb{D} \times \mathbb{U}) \times (\mathbb{D} \times \mathbb{U})$$
$$\mathsf{port}.[\![\ a \longmapsto\!\dashv b\ ]\!] = \mathsf{fix}\,(X = ab \cdot X)$$

Note the relational, data semantics is the universal relation: any combination of input or output values is allowed (and irrelevant).

Fifo$_1$. This is a channel with a buffer of a single position. Thus $\mathbb{U} = \mathbb{D}$

$$\mathsf{data}.[\![\ a \longmapsto\!\square\!\rightarrow b\ ]\!] = \mathsf{R}_{\square}$$
$$\mathsf{port}.[\![\ a \longmapsto\!\square\!\rightarrow b\ ]\!] = \mathsf{fix}\,(X = a \cdot b \cdot X)$$

where $\mathsf{R}_{\square}$ is given by the following clauses, for all $d, u \in \mathbb{D}$,

$$(\bot, d)\ \mathsf{R}_{\square}\ (d, \bot) \quad \text{and} \quad (u, \bot)\ \mathsf{R}_{\square}\ (\bot, u)$$

The first clause corresponds to the effect of an input at port a, whereas the second captures output at port b, which requires the presence of a datum in the internal state.

Lossy channel. In a number of practical situations component orchestration depends not only on port activation, but also on the absence of service requests at particular ports in configuration. A typical example is provided by one of the basic channels in REO: the *lossy channel*, which acts as a synchronous one if both an input and an output requests are pending on its source and sink ends, respectively. However, it looses any data item on input on the absence of an output request in the other end. This behaviour is distinct from that of an *unreliable* channel, loosing data non deterministically.

To handle these cases we enrich the specification of the set of actions $\mathbb{A}$ to include *negative port activations*, or more rigorously stated, absence of port requests, denoted, for each port p, by $\tilde{p}$. Technically, actions are given by datatype

$$\mathbb{A} = \mathcal{P}(\mathbb{P} \cup \{\tau\}) \times \mathcal{P}\,\mathbb{P} \tag{4}$$

subject to the following invariant

$$\mathsf{disjoint}\langle pos, neg\rangle = (pos \cap neg = \emptyset) \tag{5}$$

Moreover, *absence of port information* is only relevant to *output* ports, which may carry, or not, a request for receiving information. Therefore, if a is an *input* port, $a = \tilde{a}$.

Values of type $\mathbb{A}$ are represented according to the following abbreviation

$$\langle \{a, b, c\}, \{d, f\} \rangle \stackrel{\text{abv}}{=} abc\tilde{d}f \tag{6}$$

Therefore, the specification of a lossy channel becomes

$$\mathsf{data}.[\![\ \bullet \cdots\!\succ \bullet\ ]\!] \subseteq \mathsf{Id}_{\mathbb{D}} \tag{7}$$
$$\mathsf{port}.[\![\ \bullet \cdots\!\succ \bullet\ ]\!] = \mathsf{fix}(X = ab \cdot X + a\tilde{b} \cdot X) \tag{8}$$

where $\tilde{b}$ corresponds to an absence of a reading request at port b. A lossy channel only transmits if a potential receiver is asking for the data item sent. Otherwise, data is lost.

III. COMPOSING CONNECTORS

Connectors can be composed through a set of six *combinators*: *parallel* and *concurrent* composition, *hook*, *interleaving*, *left join* and *right join*. This set of combinators was already introduced in [6]. In the sequel, however, the specifications of the first three are modified to correctly capture general context-awareness; the remaining are repeated for completeness of exposition and fully understanding of the propagation examples in section IV.

1) Aggregation: There are three combinators, denoted by $\boxplus$, $\boxtimes$ and $\boxdot$, whose effect is to place their arguments side-by-side. The distinction is related to the way they combine the behavioural patterns: through *parallel* composition or *interleaving*, respectively. Formally,

$$\mathsf{port}.[\![\mathbb{C}_1 \boxplus \mathbb{C}_2]\!] = \mathsf{port}.[\![\mathbb{C}_1]\!] \boxplus \mathsf{port}.[\![\mathbb{C}_2]\!]$$
$$\mathsf{port}.[\![\mathbb{C}_1 \boxtimes \mathbb{C}_2]\!] = \mathsf{port}.[\![\mathbb{C}_1]\!] \boxtimes \mathsf{port}.[\![\mathbb{C}_2]\!]$$
$$\mathsf{port}.[\![\mathbb{C}_1 \boxdot \mathbb{C}_2]\!] = \mathsf{port}.[\![\mathbb{C}_1]\!] \boxdot \mathsf{port}.[\![\mathbb{C}_2]\!]$$

taking, in the first two cases, $\theta = \cup$ to capture the envisaged interaction discipline.

At data level all combinators behave as a relational product. Formally, for $\square = \boxplus, \boxtimes, \boxdot$,

$$\mathsf{data}.[\![\mathbb{C}_1 \square \mathbb{C}_2]\!] = \mathsf{data}.[\![\mathbb{C}_1]\!] \mathbin{\underline{\times}} \mathsf{data}.[\![\mathbb{C}_2]\!]$$

where

$$((\vec{d'},\vec{e'}),(u',v'))\ R \underline{\times} S\ ((\vec{d},\vec{e}),(u,v)) \\ \equiv\ (\vec{d'},u')\ R\ (\vec{d},u)\ \wedge\ (\vec{e'},v')\ S\ (\vec{e},v)$$

Combinators $\boxtimes$ and $\boxdot$ admit *strong* versions in order to propagate negative information. They are denoted by $\otimes$ and $\odot$, respectively.

Let us consider first $\otimes$, the strong synchronous product. The intuition underlying the definition of $P \otimes Q$ is as follows: whenever a term in the expansion of P has a negative port $\tilde{p}$ (and recall that by (**??**) p has an *output* polarity), it must be *multiplied* by $\tilde{P}$ to prepare the grounds for propagating the associated negative information. This may, in particular, turn *negative* an output port in P which may represent the propagation of negative information. Examples will be given soon after the introduction of the *hook* combinator. Formally, let $\Upsilon(P)$ denote the immediate expansion of behavioural expression P, *i.e.*,

$$\Upsilon(P)\ =\ \sum_{i\in\{1,\cdots,m\}} \omega_i \cdot P_i \tag{9}$$

such that $P \sim \Upsilon(P)$, where $\sim$ stands for bisimilarity. Each summand F in $\Upsilon(P)$ is said to be a factor of P, a fact we represent by $F \leftarrow \Upsilon(P)$. Let $F = \omega \cdot R$ such that

$$\omega \cdot R \leftarrow \Upsilon(P)\ \text{ and }\ \omega \cap \widetilde{\mathbb{A}} \neq \emptyset$$

In this case F is said to be a factor of P with negated ports and represented by $F \leftarrow \Upsilon_n(P)$. Now define $\otimes$ as

$$\otimes\ =\ [\![\alpha_\otimes]\!] \tag{10}$$

with

$$\alpha_\otimes\,(P,Q)\ = \alpha_\boxtimes\,(P,Q) \cup \bigcup_{\substack{\omega\cdot P'\leftarrow\Upsilon_n(P)\\ j\in\{1,\cdots,n\}}} (\omega\cup\widetilde{\omega_j},(P',Q_j)) \\ \cup \bigcup_{\substack{\omega\cdot Q'\leftarrow\Upsilon_n(Q)\\ i\in\{1,\cdots,m\}}} (\widetilde{\omega_i}\cup\omega,(P_i,Q'))$$

for $\Upsilon(P) = \sum_{i\in\{1,\cdots,m\}}\ \omega_i\cdot P_i$ and $\Upsilon(Q) = \sum_{j\in\{1,\cdots,n\}}\ \omega_j\cdot Q_j$.

Note that (10) corresponds to the following explicitly recursive definition:

$$P\otimes Q\ =\ P\boxtimes Q + \sum_{\substack{\omega\cdot P'\leftarrow\Upsilon_n(P)\\ j\in\{1,\cdots,n\}}} (\omega\cup\widetilde{\omega_j})\cdot(P'\otimes Q_j) \\ + \sum_{\substack{\omega\cdot Q'\leftarrow\Upsilon_n(Q)\\ i\in\{1,\cdots,m\}}} (\widetilde{\omega_i}\cup\omega)\cdot(P_i\otimes Q') \tag{11}$$

Clearly, if neither P nor Q have negative factors $P\otimes Q = P\boxtimes Q$.

The strong version of parallel composition is defined by combining, at the *genes* level, the effects of both $\boxplus$ and $\otimes$, just as $\boxdot$ combines $\boxplus$ and $\boxtimes$. Formally,

$$\odot\ =\ [\![\alpha_\odot]\!]$$

where

$$\alpha_\boxdot\ =\ \nu\times\nu \xrightarrow{\cup\cdot(\alpha_\boxplus\times\alpha_\otimes)\cdot\Delta} \mathcal{P}\,(\mathbb{A}\times(\nu\times\nu))$$

2) Hook: This combinator encodes a *feedback* mechanism, drawing a direct connection between an output and an input port. This has a double consequence: the connected ports must be activated simultaneously and become externally non observable. Formally, such conditions must be expressed in $\mathsf{port}.[\![\mathbb{C}\curvearrowleft_i^j]\!]$ and their specification requires some care.

The crucial issue is the suitable definition of a new combinator for behaviours, $\mathsf{hide}\,c$, parametric on a set $c\subseteq\mathbb{A}-\{0\}$, whose effect is to prune its argument according to the following rules:

- all computations exhibiting occurrences of non empty strict subsets of c must be removed, because ports in c have be activated simultaneously;
- there is, however, an exception to the rule above: if a computation exhibits a non empty strict subset c' of c such that c' only contains negative *output* ports, a property denoted by $\mathsf{negfac}(c')$, then such a computation is not removed.

The intuition for the last rule is that if the only occurrence of an *output* port which the hook combinator aims to internalise, is negative, *i.e.*, has, in the computation considered, no pending output request, it can be ignored: there is no matching input port, but also no information to be transmitted.

The combinator then hides all references to c in the remaining computations, either by removing them when occurring in a strictly larger context or by mapping them to an unobservable action τ when occurring isolated. It is defined as

$$\mathsf{hide}\,c\ =\ [\![\alpha_{\mathsf{hide}\,c}]\!] \tag{12}$$

where

$$\alpha_{\mathsf{hide}\,c}\ =\ \nu\xrightarrow{\omega}\mathcal{P}\,(\mathbb{A}\times\nu)\xrightarrow{\mathsf{h}_c}\mathcal{P}\,(\mathbb{A}\times\nu)$$

and

$$\mathsf{h}_c\ s\ = \{\langle a\setminus c,u\rangle\ \mid\ \langle a,u\rangle\in s\ \wedge \\ ((a\cap c\neq\emptyset)\ \rightarrow\ (c\subset a\ \vee\ \mathsf{negfac}(a\cap c)))\} \\ \cup\ \{\langle\tau,u\rangle|\ \langle c,u\rangle\in s\}$$

Thus, let i, respectively j, be an output, respectively, input, port in connector $\mathbb{C}$. The *hook* combinator links i to j according to the following definition:

$$\mathsf{port}.[\![\mathbb{C}\curvearrowleft_i^j]\!]\ =\ \mathsf{hide}\,\{i,j\}\,\mathsf{hide}\,\{\widetilde{i},\widetilde{j}\}\,\mathsf{port}.[\![\mathbb{C}]\!]$$

If $\mathsf{data}.[\![\mathbb{C}]\!] : \mathbb{D}^n\times U \longleftarrow \mathbb{D}^m\times U$, the effect of *hook* on the data flow relation is modelled by relation

$$\mathsf{data}.[\![\mathbb{C}\curvearrowleft_i^j]\!] : \mathbb{D}^{n-1}\times U \longleftarrow \mathbb{D}^{m-1}\times U$$

$$t'_{|j}\,(\mathsf{data}.[\![\mathbb{C}\curvearrowleft_i^j]\!])\,t_{|i}\quad\text{iff}\quad t'\,(\mathsf{data}.[\![\mathbb{C}]\!])\,t\ \wedge\ t'_{\#j}=t_{\#i}$$

Example 1: Let us illustrate the hook combinator through an elementary example, which does not involve negative information. More complex examples are discussed in next section, in which it is shown the suitability of this combinator to handle composition of context-aware connectors. For the moment, consider connectors $\mathbb{C}$ and $\mathbb{F}$, both with an input and an output port, named a, a' in the first case, and b, b' in the second. Let us analyse composition $(\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'}$. At the data level, one gets

$$\begin{array}{ll} & (y,(u',v'))\ \mathsf{data}.[\![(\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'}]\!]\ (x,(u,v)) \\ = & \quad \{\ \text{unfolding definitions}\ \} \\ & \exists_z\ .\ \ ((z,y),(u',v'))\ \mathsf{data}.[\![\mathbb{C} \odot \mathbb{F}]\!]\ (x,z),(u,v)) \\ = & \quad \{\ \text{unfolding definitions}\ \} \\ & \exists_z\ .\ \ (z,u')\ \mathsf{data}.[\![\mathbb{C}]\!]\ (x,u)\ \wedge\ (y,v')\ \mathsf{data}.[\![\mathbb{F}]\!]\ (z,v) \end{array}$$

which shows that the *hook combinator* encodes a form of relational composition which is *partial* in the sense that only part of the output is fed back as new input. For the behavioural part, consider $\mathbb{C}$ and $\mathbb{F}$ as synchronous channels. Then,

$$\mathsf{port}.[\![(\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'}]\!]\ =\ \mathsf{fix}\ (x = ab'.x)$$

because the other two terms in the expansion $\mathsf{fix}\ (x = aa'.x + bb'.x + aa'bb'.x)$ contain strict subsets of $c = \{a', b\}$. Note that the synchronous channel always acts as the identity for *hook*. Suppose, now, that $\mathbb{F}$ is defined as a $Fifo_1$ channel. Thus, and adopting, in the sequel, the convention which abbreviates $\mathsf{port}.[\![\mathbb{C}]\!]$ to $\mathbb{C}$,

$$\begin{array}{ll} & \mathsf{port}.[\![(\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'}]\!] \\ \sim & \quad \{\ \text{hook definition and expansion law}\} \\ & aa'b \cdot (\mathbb{C} \odot b' \cdot \mathbb{F}) \curlywedge^{b}_{a'} + aa' \cdot (\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'} \\ & + b \cdot (\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'} \\ \sim & \quad \{\ \mathsf{hide}\ \ \text{definition}\ \} \\ & a \cdot (\mathbb{C} \odot b' \cdot \mathbb{F}) \curlywedge^{b}_{a'} \\ \sim & \quad \{\ \text{expansion law}\ \} \\ & a \cdot (aa' \cdot (\mathbb{C} \odot b' \cdot \mathbb{F}) \curlywedge^{b}_{a'} + b' \cdot (\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'} \\ & + aa'b' \cdot (\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'}) \\ \sim & \quad \{\ \mathsf{hide}\ \ \text{definition}\ \} \\ & a \cdot b' \cdot (\mathbb{C} \odot \mathbb{F}) \curlywedge^{b}_{a'} \\ \sim & \quad \{\ \text{introducing fix}\ \} \\ & \mathsf{fix}\ (x = a \cdot b' \cdot x) \end{array}$$

Hint *expansion law* used in the derivation above refers to a basic result in process algebra, although often presented in different formulations: *a process is bisimilar to the non deterministic choice of its immediate derivatives*.

3) Join: The last combinator considered here is called *join* and its effect is to plug ports with identical polarity. The aggregation of *output* ports is done by a *right join* $(\mathbb{C}\ {}^{i}_{j} > z)$, where $\mathbb{C}$ is a connector, i and j are ports and z is a fresh name used to identify the new port. Port z receives asynchronously messages sent by either i or j. When messages are sent at same time the combinator chooses one of them non deterministically.

On the other hand, aggregation of *input* ports resorts to a *left join* $(z <^{i}_{j} \mathbb{C})$. This behaves like a $broadcaster$ sending synchronously messages from z to both i and j. Formally, for $\mathsf{data}.[\![\mathbb{C}]\!] : \mathbb{D}^n \times U \longleftarrow \mathbb{D}^m \times U$, we define

Right join:
The data flow relation $\mathsf{data}.[\![\mathbb{C}\ {}^{i}_{j} > z]\!] : \mathbb{D}^{n-1} \times U \longleftarrow \mathbb{D}^m \times U$ for this operator is given by $r\ (\mathsf{data}.[\![\mathbb{C}\ {}^{i}_{j} > z]\!])\ t$ iff

$$t'\ (\mathsf{data}.[\![\mathbb{C}]\!])\ t\ \wedge\ r_{|z} = t'_{|i,j}\ \wedge\ (r_{\#z} = t'_{\#i}\ \vee\ r_{\#z} = t'_{\#j})$$

Notation $t_{\#i}$ stands for the component of data tuple t corresponding to port i. On the other hand, $t_{|i}$ as a tuple identical to t from which component $t_{\#i}$ has been deleted. At the behavioural level, its effect is that of a renaming operation

$$\mathsf{port}.[\![(\mathbb{C}\ {}^{i}_{j} > z)]\!]\ =\ \{z \leftarrow i, z \leftarrow j\}\ \mathsf{port}.[\![\mathbb{C}]\!]$$

Example 2: The merger connector depicted in Figure 2 is obtained by a right join of the sink end of two interleaved synchronous channels.

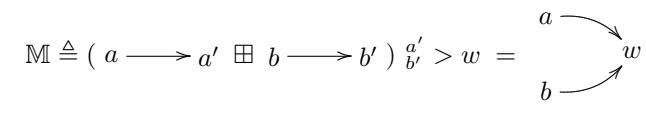


Fig. 2. A *merger*.

Its behavioural pattern is

$$\mathsf{port}.[\![\mathbb{M}]\!]\ =\ \mathsf{fix}\ (x = aw.x + bw.x)$$

because,

$$\begin{array}{ll} & \mathsf{port}.[\![\mathbb{M}]\!] \\ \sim & \quad \{\ \text{definition of right join and synchronous channel}\ \} \\ & \{w \leftarrow a', w \leftarrow b'\}\ (\mathsf{fix}\ (x = aa' \cdot x)\ \boxplus\ \mathsf{fix}\ (x = bb' \cdot x)) \\ \sim & \quad \{\ \text{definition of interleaving and expansion law}\ \} \\ & \{w \leftarrow a', w \leftarrow b'\}\ (\mathsf{fix}\ (x = aa' \cdot x\ +\ bb' \cdot x\ )) \\ \sim & \quad \{\ \text{substitution}\ \} \\ & \mathsf{fix}\ (x = aw.x + bw.x) \end{array}$$

Left join:
The behaviour of a left join is a little more complex: before renaming, all computations of $\mathbb{C}$ in which ports i and j are activated independently of each other must be removed. Again this is specified by a new process combinator $\mathsf{force}\, c$ which forces the joint activation of a set c of ports. Formally,

$$\mathsf{force}\, c\ =\ [\![\alpha_{\mathsf{force}\, c}]\!] \tag{13}$$

where

$$\begin{array}{rl} \alpha_{\mathsf{force}\, c} & = \nu \xrightarrow{\omega} \mathcal{P}\,(\mathbb{A} \times \nu) \xrightarrow{\mathsf{f}_c} \mathcal{P}\,(\mathbb{A} \times \nu) \\ \mathsf{f}_c\ s & = \{\langle a, u\rangle \in s |\ a \cap c\ \subseteq\ \{\emptyset, c\}\} \end{array}$$

Thus □

$$\mathsf{port}.[\![(z <^i_j \mathbb{C})]\!] \;=\; \{z \leftarrow i, z \leftarrow j\}\ \mathsf{force}\,\{i,j\}\ \mathsf{port}.[\![\mathbb{C}]\!]$$

On the other hand, the data flow specification is given by $t'\,(\mathsf{data}.[\![z <^i_j \mathbb{C}]\!])\,r$ iff

$$t'\,(\mathsf{data}.[\![\mathbb{C}]\!])\,t \;\wedge\; r_{|z} = t_{|i,j} \;\wedge\; r_{\#z} = t_{\#i} = t_{\#j}$$

IV. Propagation of context dependent behaviour

The study of context dependent behaviour, and its propagation by composition, in exogenous coordination models was motivated by the behaviour of channels which react differently depending on the presence or absence of information at their ports. The prototypical case is the Reo *lossy* channel defined, in our formalism, by (7) and (8). In this section we show that the proposed model, with *negative* information, is able the pass two tests which constitute the hallmark of propagation of context dependent behaviour. They are concerned with the behaviour of a *lossy* channel composed either with a synchronous channel or an empty *fifo*$_1$. We formulate then as two lemmas to sustain our claim.

Lemma 1: Whenever a *lossy* channel is composed (via $\odot$ and *hook*) with a synchronous channel, on either side, the result must be again a *lossy* channel, *i.e.*

$$(\ a \cdots\!\!\succ a'\ \ \odot\ \ b' \longmapsto b\ )\ {}^{\leftarrow b'}_{a'} \;=\; a \cdots\!\!\succ b \qquad (14)$$

$$(\ a \longmapsto a'\ \ \odot\ \ b' \cdots\!\!\succ b\ )\ {}^{\leftarrow b'}_{a'} \;=\; a \cdots\!\!\succ b \qquad (15)$$

Proof. We concentrate on the behaviour part. For the data component of the definition just observe that the identity relation is always the identity for relational composition. Thus, for (14),

$$\begin{aligned}
&\mathsf{port}.[\![(\ a \cdots\!\!\succ a'\ \ \odot\ \ b' \longmapsto b\ )\ {}^{\leftarrow b'}_{a'}]\!]\\
\sim\ &\{\text{ definition of } \odot \text{ and hook}\}\\
&\mathsf{fix}\ (X = aa'\cdot X + a\tilde{a'}\cdot X + aa'bb'\cdot X + a\tilde{a'}bb'\cdot X\\
&+bb'\cdot X + a\tilde{a'}\tilde{b}\tilde{b'}\cdot X)\\
\sim\ &\{\text{ definition of } \mathsf{hide}\ \}\\
&\mathsf{fix}\ (X = ab\cdot X + a\tilde{b}\cdot X)\\
\sim\ &\{\text{ definition of a } \textit{lossy} \text{ channel}\}\\
&\mathsf{port}.[\![\ a \cdots\!\!\succ b\ ]\!]
\end{aligned}$$

Similarly, for (15),

$$\begin{aligned}
&\mathsf{port}.[\![(\ a \longmapsto a'\ \ \odot\ \ b' \cdots\!\!\succ b\ )\ {}^{\leftarrow b'}_{a'}]\!]\\
\sim\ &\{\text{ definition of } \odot \text{ and hook, expansion law}\}\\
&\mathsf{fix}\ (X = aa'\cdot X + bb'\cdot X + \tilde{b}b'\cdot X + aa'bb'\cdot X\\
&+aa'\tilde{b}b'\cdot X + \tilde{a}a'b\tilde{b'}\cdot X)\\
\sim\ &\{\text{ definition of } \mathsf{hide}\ \}\\
&\mathsf{fix}\ (X = ab\cdot X + a\tilde{b}\cdot X + \tilde{a}b\cdot X)\\
\sim\ &\{\text{ property (??), + idempotent}\}\\
&\mathsf{fix}\ (X = ab\cdot X + a\tilde{b}\cdot X)\\
\sim\ &\{\text{ definition of a } \textit{lossy} \text{ channel}\}\\
&\mathsf{port}.[\![\ a \cdots\!\!\succ b\ ]\!]
\end{aligned}$$

Lemma 2: Whenever a *fifo*$_1$ is composed on the right with a *lossy* channel data must flow through the latter to the former as the input port of an empty *fifo*$_1$ is always presenting a reading request. Therefore, no data is lost. Formally,

$$(\ a \cdots\!\!\succ a'\ \ \odot\ \ b' \longmapsto\!\!\square\!\!\succ b\ )\ {}^{\leftarrow b'}_{a'} \;=\; a \longmapsto\!\!\square\!\!\succ b \qquad (16)$$

Proof. Let $L = \mathsf{port}.[\![\ a \cdots\!\!\succ a'\ ]\!]$ and $F = \mathsf{port}.[\![\ b' \longmapsto\!\!\square\!\!\succ b\ ]\!]$. Then

$$\begin{aligned}
&\mathsf{port}.[\![(\ a \cdots\!\!\succ a'\ \ \odot\ \ b' \longmapsto\!\!\square\!\!\succ b\ )\ {}^{\leftarrow b'}_{a'}]\!]\\
\sim\ &\{\text{ definition of channels, } \odot \text{ and hook, expansion law}\}\\
&aa'\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'} + a\tilde{a'}\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'} +\\
&b'\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'} + aa'b'\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'}\\
&+a\tilde{a'}b'\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'} + a\tilde{a'}\tilde{b'}\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'}\\
\sim\ &\{\text{ property (??), + idempotent}\}\\
&aa'\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'} + a\tilde{a'}\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'} +\\
&b'\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'} + aa'b'\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'} +\\
&+a\tilde{a'}\tilde{b'}\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'}\\
\sim\ &\{\text{ definition of } \mathsf{hide}\ \}\\
&a\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'} + a\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'}\\
\sim\ &\{\text{ + idempotent}\}\\
&a\cdot(L\odot b\cdot F)\ {}^{\leftarrow b'}_{a'}\\
\sim\ &\{\text{ definition of channels, } \odot \text{ and hook, expansion law}\}\\
&a\cdot(aa'\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'} + a\tilde{a'}\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'} +\\
&b\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'} + a\tilde{a'}b\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'} +\\
&a\tilde{a'}\tilde{b}\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'})\\
\sim\ &\{\text{ definition of } \mathsf{hide}\ \}\\
&a\cdot b\cdot(L\odot F)\ {}^{\leftarrow b'}_{a'}\\
\sim\ &\{\text{ introducing fix}\}\\
&\mathsf{fix}\ (X = a\cdot b\cdot X)\\
\sim\ &\{\text{ definition of a } \textit{fifo}_1 \text{ channel}\}\\
&\mathsf{port}.[\![\ a \longmapsto\!\!\square\!\!\succ b\ ]\!]
\end{aligned}$$

For the data component notice that, once the *lossy* channel never looses any data, as just shown, its static semantics, in this particular composition, is the identity relation. Therefore

$$\mathsf{data}.[\![\ a \cdots\!\!\succ a'\ \ \odot\ \ b' \longmapsto\!\!\square\!\!\succ b\ )\ {}^{\leftarrow b'}_{a'}]\!] \;=\; \mathsf{data}.[\![\ a \longmapsto\!\!\square\!\!\succ b\ ]\!]$$

□

Lemmas 1 and 2 establish the adequacy of this model to propagate context dependent behaviour. It is also instructive to compute the joint behaviour of a *fifo*$_1$ with a *lossy* channel. This yields, for $F = \mathsf{port}.[\![\ a \longmapsto\!\!\square\!\!\succ a'\ ]\!]$ and $L = \mathsf{port}.[\![\ b' \cdots\!\!\succ b\ ]\!]$,

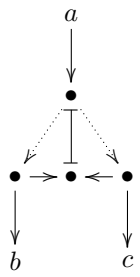


Fig. 3. XR — The *exclusive router* connector.

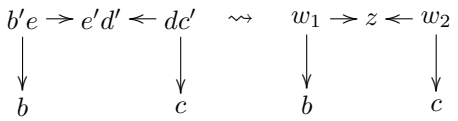
$$
\begin{array}{ll}
 & \mathsf{port}.[\![(\ a \vdash\!\square\!\succ a' \ \ \odot \ \ b' \cdots\!\succ b\ )\ {}^{\backslash b'}_{a'}]\!] \\
\sim & \quad \{\ \text{definition of channels, } \odot \text{ and hook, expansion law}\} \\
 & a \cdot (a' \cdot F \odot L)\ {}^{\backslash b'}_{a'} + bb' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \\
 & \tilde{b}b' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + ab\tilde{b}b' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \\
 & a\tilde{b}b' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \tilde{a}\tilde{b}b' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} \\
\sim & \quad \{\ \text{definition of hide}\ \} \\
 & a \cdot (a' \cdot F \odot L)\ {}^{\backslash b'}_{a'} \\
\sim & \quad \{\ \text{definition of channels, } \odot \text{ and hook, expansion law}\} \\
 & a \cdot (a' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + bb' \cdot (a' \cdot F \odot L)\ {}^{\backslash b'}_{a'} + \\
 & \tilde{b}b' \cdot (a' \cdot F \odot L)\ {}^{\backslash b'}_{a'} + a'bb' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \\
 & a'\tilde{b}b' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \tilde{a'}bb' \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \\
 & \tilde{a'}\tilde{b}b' \cdot (F \odot L)\ {}^{\backslash b'}_{a'}) \\
\sim & \quad \{\ \text{property (??) and definition of hide}\ \} \\
 & a \cdot (b \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \tilde{b} \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \\
 & b \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \tilde{b} \cdot (F \odot L)\ {}^{\backslash b'}_{a'}) \\
\sim & \quad \{\ +\ \text{idempotent}\} \\
 & a \cdot (b \cdot (F \odot L)\ {}^{\backslash b'}_{a'} + \tilde{b} \cdot (F \odot L)\ {}^{\backslash b'}_{a'}) \\
\sim & \quad \{\ \text{introducing fix}\} \\
 & \mathsf{fix}\ (X\ =\ a \cdot (b \cdot X + \tilde{b} \cdot X))
\end{array}
$$

V. A PARADIGMATIC EXAMPLE

To provide a 'flavour' of how the the emergent behaviour of a composed connector is computed, consider the *exclusive router* connector depicted in Fig. 3.

The intended behaviour for this connector is to transmit either in b or c, but not in both, whatever receives in a.

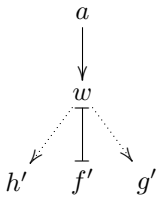
One component of XR, depicted in the lower part of the diagram, is the *right join* by e' and d' mapping to new port z, of two *broadcasters* composed by $\boxtimes$. Each broadcaster is obtained by left joining the relevant synchronous channels. The assembly process of XR_1 is represented as



The computed behavioural pattern is

$$\mathsf{port}.[\![\mathsf{XR}_1]\!]\ \sim\ \mathsf{fix}\ (X\ =\ bzw_1 \cdot X + czw_2 \cdot X)$$

The other component, XR_2 is a left join of two lossy channels and a drain, mapping their source ports, h, g and f, to w, sequentially composed with a synchronous channel from a to a', *i.e.*,



Its behavioural pattern is computed in Fig. 4.

Finally, connector XR is assembled as

$$\mathsf{XR}\ \triangleq\ (\mathsf{XR}_1 \otimes \mathsf{XR}_2)\ {}^{\backslash w_1, w_2, f'}_{h', g', w} \tag{17}$$

leading, as expected, to

$$
\begin{array}{ll}
 & \mathsf{port}.[\![\mathsf{XR}]\!] \\
\sim & \quad \{\ \text{by (17)}\} \\
 & \mathsf{hide}\{h', g', w, w_1, w_2, f'\}\ (\mathsf{port}.[\![\mathsf{XR}_1]\!] \otimes \mathsf{port}.[\![\mathsf{XR}_2]\!]) \\
\sim & \quad \{\ \text{computed above}\} \\
 & \mathsf{hide}\{h', g', w, w_1, w_2, f'\} \\
 & (\mathsf{fix}\ (X\ =\ bzw_1 \cdot X + czw_2 \cdot X)\ \otimes \\
 & \ \mathsf{fix}\ (X\ =\ ah'f'g' \cdot X + a\widetilde{h'}f'g' \cdot X + ah'f'\widetilde{g'} \cdot X + \\
 & \ af'\widetilde{h'}\widetilde{g'} \cdot X)) \\
\sim & \quad \{\ \text{definition of } \otimes\text{; expansion law}\} \\
 & \mathsf{hide}\{h', g', w, w_1, w_2, f'\} \\
 & \quad \mathsf{fix}\,(X\ =\ bzw_1ah'f'g' \cdot X + bzw_1a\widetilde{h'}f'g' \cdot X + \\
 & \qquad bzw_1ah'f'\widetilde{g'} \cdot X + bzw_1af'\widetilde{h'}\widetilde{g'} \cdot X + \\
 & \qquad czw_2ah'f'g' \cdot X + czw_2a\widetilde{h'}f'g' \cdot X + \\
 & \qquad czw_2ah'f'\widetilde{g'} \cdot X + czw_2af'\widetilde{h'}\widetilde{g'} \cdot X + \\
 & \qquad \widetilde{bz}\widetilde{w_1}ah'f'g' \cdot X + \widetilde{bz}\widetilde{w_1}a\widetilde{h'}f'g' \cdot X + \\
 & \qquad \widetilde{bz}\widetilde{w_1}af'\widetilde{h'}\widetilde{g'} \cdot X + \widetilde{cz}\widetilde{w_2}a\widetilde{h'}f'g' \cdot X + \\
 & \qquad \widetilde{cz}\widetilde{w_2}ah'f'\widetilde{g'} \cdot X + \widetilde{cz}\widetilde{w_2}af'\widetilde{h'}\widetilde{g'} \cdot X) \\
\sim & \quad \{\ \text{definition of hide}\} \\
 & \mathsf{fix}\,(X\ =\ ab \cdot X + ab \cdot X + ac \cdot X + ac \cdot X + \\
 & a\widetilde{b} \cdot X + a\widetilde{b} \cdot X + a\widetilde{c} \cdot X + a\widetilde{c} \cdot X) \\
\sim & \quad \{\ +\ \text{idempotent}\} \\
 & \mathsf{fix}\ (X\ =\ ab \cdot X + ac \cdot X + a\widetilde{b} \cdot X + a\widetilde{c} \cdot X)
\end{array}
$$

Notice that the application of hide in the calculation above made use of the possibility of keeping terms where the intersection of their prefix set of ports with the argument of hide reduce to a *negated output* port (cases of $\widetilde{h'}$ and $\widetilde{g'}$).

$$
\begin{aligned}
& \mathsf{port}.[\![\mathsf{XR}_2]\!] \\
\sim \quad & \{ \text{ definiton of } \mathsf{XR}_2 \} \\
& \mathsf{port}.[\![(\ a \longrightarrow a \ \odot w <_r^f \ r <_g^h \ (\ h \cdots\cdots h' \ \otimes \ f \longmapsto f' \ \otimes \ g \cdots\cdots g' \ )) \ \curvearrowleft_{a'}^{w}]\!] \\
\sim \quad & \{ \text{ definitions of channels, } \otimes \text{, left join and hook; expansion law } (\mathbf{??}) \} \\
& \mathsf{hide}\{a', w\} \, (\mathsf{fix} \ (X \ = \ aa' \cdot X) \ \odot \\
& \qquad \mathsf{fix} \ (X \ = \ wh'f'g' \cdot X + w\widetilde{h'}f'g' \cdot X + wh'f'\widetilde{g'} \cdot X + wf'\widetilde{h'}\widetilde{g'} \cdot X)) \\
\sim \quad & \{ \text{ definition of } \odot \} \\
& \mathsf{hide}\{a', w\} \, \mathsf{fix} \, (X \ = \ aa' \cdot X + wh'f'g' \cdot X + w\widetilde{h'}f'g' \cdot X + wh'f'\widetilde{g'} \cdot X + wf'\widetilde{h'}\widetilde{g'} \cdot X \\
& \qquad + aa'wh'f'g' \cdot X + aa'w\widetilde{h'}f'g' \cdot X + aa'wh'f'\widetilde{g'} \cdot X + aa'wf'\widetilde{h'}\widetilde{g'} \cdot X \\
& \qquad + \widetilde{a}a'wh'f'g' \cdot X + \widetilde{a}a'w\widetilde{h'}f'g' \cdot X + \widetilde{a}a'wh'f'\widetilde{g'} \cdot X + \widetilde{a}a'wf'\widetilde{h'}\widetilde{g'} \cdot X) \\
\sim \quad & \{ \text{ definition of } \mathsf{hide} \} \\
& \mathsf{fix} \ (X \ = \ ah'f'g' \cdot X + a\widetilde{h'}f'g' \cdot X + ah'f'\widetilde{g'} \cdot X + af'\widetilde{h'}\widetilde{g'} \cdot X)
\end{aligned}
$$

Fig. 4. Computing the behaviour of XR_2 .

VI. Conclusion

A model for context-aware software connectors was proposed, improving on [6], and checked against typical context propagation requirements. The model is based on separate data and behaviour specifications, the later resorting to a 'tailor-made' process algebra.

Although the notion of a software connector used in this paper was borrowed from Reo, the underlying semantic principles are largely independent of a particular coordination language. With respect to the former, however, it should be noted that the explicit use of combinators provides a structural alternative to composition through graph manipulation, used in mainstream Reo literature. It also entails proofs by standard equational reasoning, even avoiding the explicit construction of bisimulations.

Of course, a lot of work remains to be done. In particular we are working on developing notions of connector equivalence and refinement on top of which a basis for a connector calculus could be studied.

References

[1] F. Arbab. Abstract Behavior Types: A Foundation Model for Components and Their Composition. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Formal Methods for Components and Objects: First International Symposium, FMCO 2002, Leiden, The Netherlands, November 2002, Revised Lectures*, volume 2852 of *LNCS*, pages 33–70. Springer, 2003.

[2] F. Arbab. Reo: A Channel-based Coordination Model for Component Composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[3] F. Arbab and J. J. M. M. Rutten. A coinductive calculus of component connectors. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques, 16th Inter. Workshop, WADT 2002, Revised Selected Papers*, pages 34–55. Springer Lect. Notes Comp. Sci. (2755), 2003.

[4] C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in reo by constraint automata. *Science of Computer Programming*, 61(2):75–113, 2006.

[5] L. S. Barbosa. Process calculi *à la* Bird-Meertens. In M. L. Andrea Corradini and U. Montanari, editors, *CMCS'01*, volume 44.4, pages 47–66, Genova, April 2001. Elect. Notes in Theor. Comp. Sci., Elsevier.

[6] M. A. Barbosa and L. S. Barbosa. A perspective on service orchestration. *Science of Computer Programming*, 74(9):671–687, 2009.

[7] M. Bonsangue, D. Clarke, and A. Silva. Automata for context-dependent connectors. In J. Field and V. Vasconcelos, editors, *Proc. Coordiantion 2009*, volume 5521 of *Lecture Notes in Computer Science*, pages 184–203. Springer Verlag, 2009.

[8] D. Clarke, D. Costa, and F. Arbab. Connector colouring I: Synchronisation and context dependency. *Science of Computer Programming*, 66(3):205–225, 2007.

[9] D. Costa. *Formal models for context dependent connectors for distributed software components and services (forthcoming PhD thesis)*. PhD thesis, Vrije Universiteit Amsterdam, 2010.

[10] J. L. Fiadeiro. Software services: scientific challenge or industrial hype? In K. Araki and Z. Liu, editors, *Proc. First International Colloquim on Theoretical Aspects of Computing (ICTAC'04), Guiyang, China*, pages 1–13. Springer Lect. Notes Comp. Sci. (3407), 2004.

[11] J. L. Fiadeiro and A. Lopes. Semantics of Architectural Connectors. In M. Bidoit and M. Dauchet, editors, *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Lille, France, April 14-18, 1997, Proceedings*, volume 1214 of *LNCS*, pages 505–519. Springer-Verlag, 1997.

[12] D. Kitchin, W. R. Cook, and J. Misra. A language for task orchestration and its semantic properties. In C. Baier and H. Hermanns, editors, *Proc. 17th Inter. Conf. Concurrency Theory, CONCUR 2006, Bonn, Germany, August 27-30*, pages 477–491. Springer Lect. Notes Comp. Sci. (4137), 2006.

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

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

[15] P. Ribeiro, M. A. Barbosa, and L. S. Barbosa. Generic process algebra: A programming challenge. *Journal of Universal Computer Science*, 12(7):922–937, 2006.

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

[17] S. Stephen Kell. Rethinking software connectors. In *SYANCO '07: Inter. on Synthesis and Analysis of Component Connectors*, pages 1–12, New York, NY, USA, 2007. ACM.