Configurations of Web Services

Marco Antonio Barbosa [1]

DI-CCTC – Universidade do Minho
Braga, Portugal

Luís Soares Barbosa [2]

DI-CCTC – Universidade do Minho
Braga, Portugal

Abstract

The quest for sound foundations for the orchestration of web services is still open. To a great extent its relevance comes from the possibility of defining formal semantics for new language standards (like `BPEL4WS` or `WS-CDL`) in this emerging and challenging technology. As a step in that direction, this paper resorts to a notion of *configuration*, developed by the authors in the context of a Reo-like exogenous coordination model for software components, to formally express service orchestration. The latter is regarded as involving both the architectural assembly of independent services and the description of their interactions.

Key words: Web services, configuration, coordination.

1 Introduction

As the most popular technology in the emerging paradigm of service-oriented computation, web services are re-shaping the Web from a document-centered to a service-centered environment. The impact of such a move, both in the world's economy and in our everyday life, is just beginning to loom.

Technically the definition of what a web service is offers no special difficulty. According to the *World Wide Web Consortium*, it is just a software application identified by a uniform resource identifier (`URI`), whose interfaces and binding can be defined, described, and discovered by `XML` artifacts, and that supports direct interactions with other software application using `XML` based messages via Internet-based protocols. A sightly more refined specification would abstract from concrete representations of data and messages,

[1] Email: `marco.antonio@di.uminho.pt`
[2] Email: `lsb@di.uminho.pt`

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

defining the underlying notion of a *service* as a state-distributed and platform-independent computational entity which can be defined, published, classified, discovered and dynamically assembled for developing massively distributed, interoperable, evolvable systems and applications. Services, typically running in different platforms and owned by different organizations, interact and cooperate to achieve some complex goals. Therefore, suitable formal models for interaction and cooperation become essential to represent and reason about web service composition. Such is the theme of this paper.

In practice, web service composition is described in terms of either *choreography* or *orchestration* languages. The former specifies the conversation rules which govern interactions between all the services involved in a particular application, whereas the latter provides means to program a specific service, called the orchestrator, responsible for some form of external coordination of the services. These two approaches have been separately developed by industrial consortia and international organizations such as W3C and OASIS. In particular, `WS-CDL` and `BPEL4WS` specifications represent the most credited languages for the Web Services technology which deal with choreography and orchestration respectively.

In such a context, the starting point of this paper is the striking similarity between the orchestration service, mentioned above, and what is called *the glue code* in classical coordination approaches [24,23]. The corresponding research question is:

- Taking, as an underlying assumption, that web services do not interact directly, can a 'general-purpose' coordination model be used to specify their orchestration in typical applications?

In particular, the paper tackles this question in the context of a variant to Reo exogenous coordination model [5], developed by the authors and documented in [10,9].

This model is based on a notion of *software connector* which regulates the flow of data by relating data items present to its input and output ports. Typically the coordinated entities are regarded as black-boxes, characterized by a set of ports through which data values are sent or received. Ports have a polarity (either *input* or *output*) and, maybe, a type to classify the admissible values.

This is however clearly insufficient to count as an *interface* for a web service. The latter should also include a description of what is commonly called the service *workflow patterns*. I.e., a specification of which, when and under what conditions ports become activated (*i.e.*, ready to deliver or consume a datum). This raises a second question to be addressed in this paper:

- How can the model be extended with service interfaces exhibiting some form of behavioural specification to model the intended workflow, or *use*, pattern?

Clearly, to be useful such descriptions have to be *compositional*, in the sense that the overall behaviour of a web service application should be computed from the behaviour of individual services and that of the *connectors* forming the orchestration layer.

The paper resorts to process algebra to build such specifications. Therefore, both service interfaces and software connectors become equipped with a *use* pattern given as an expression in a process algebra. The idea is, in itself, not new. For example, reference [28] uses a process language to describe the message exchange between web services, and to *reason* about them. A similar work, but now in the choreography side, is reported in [13].

The challenging issue is composition. Actually it comes with no surprise that the interaction discipline which governs web services integration is distinct from the one underlying the global composition of web services and the *glue code*. Typical process algebras, however, have a specific interaction discipline which is fixed once and for all (*e.g.*, the action/coaction synchronization which characterizes CCS [20]).

This leads us to another piece of previous work (documented in [7,8]) on the development of *generic* process algebra. I.e., process algebras in which parallel composition is parametric on an interaction discipline suitably encoded. One may then have *different* interaction models governing different aspects of a specification.

Such is the genesis of the orchestration model proposed in this paper. Expressive power and the possibility of computing the overall behaviour of a particular web service based application are, in our opinion, its merits.

The paper results are presented in two main sections: section 2 which introduces *behavioural interfaces* for web services and section 3 which discusses their exogenous orchestration through external *glueing code*. Finally, section 4 presents some conclusions and directions for future work.

2 Behavioural Interfaces

2.1 Defining Interfaces

As mentioned before, in exogenous coordination models, like [5] or [9], components are black box entities accessed by purely syntactic interfaces. The role of an interface is restricted to keeping track of port names and, possibly, of admissible types for data items flowing through them [3]. For a web-service,

[3] In the sequel, however, we assume a unique, general data domain, denoted by $\mathbb{D}$, as the type of all data values flowing in a web service based application.

however, the specification of the corresponding workflow pattern is as important as the description of the available actions or of the orchestration structure. This leads to the following definition:

Definition 2.1 A web-service S interface is specified by a *port signature*, $sig(S)$ over $\mathbb{D}$, given by a port name and a polarity annotation (either in(put) or out(put)), and a *use pattern*, $use(S)$, given by a process term, as detailed below, over port names.

The relevant question concerns what sort of formalism should be used for the specification of *use patterns*? Transition systems [19,29], regular-expressions [25,26,31] or process algebras [18,3] are part of the huge diversity of formal structures typically used to represent behaviour, which has also been explored in the formalization of web services. Process algebra, in particular, provides an expressive setting for representing behavioural patterns and establish/verify their properties in a compositional way. Some flexibility, however, is required with respect to the underlying interaction discipline. Actually, different such disciplines have to be used, at the same time, to capture different aspects of web services orchestration. For example the discipline governing to composition of *software connectors* between them (to build the overall *glue code*) differs from the one used to capture the interaction between the latter and the relevant web services' interfaces. In any case, one needs a way of specifying the relevant interaction discipline while guaranteeing that behaviour combinators used are *parametric* on it. Meeting this goal entails the need for a *generic* way to *design* process algebras. Our previous work on a coalgebraic reconstruction of classical process calculi, documented in [7,8], provides the necessary ingredients. This work is briefly reviewed in the following sub-section, which paves the way to the discussion of its application in the context of web service orchestration.

2.2 Generic Process Algebra

References [7,8] introduced a denotational approach to the *design* of process algebras in which processes are identified with inhabitants of a final coalgebra [17] and their combinators defined by coinductive extension (of 'one-step' behaviour generator functions). The *universality* of such constructions entails both definitional and proof principles on top of which the development of the whole calculus is based [4].

[4] Combined with the *pointfree* 'calculational' style entailed by category theory, this leads to a generic way of reasoning about processes in which, in particular, proofs by bisimulation, which classically involve the explicit construction of such a relation [20], are replaced by *equational reasoning*. In the *dual* world of functional programming the role of such 'universals' is the basis of a whole discipline of algorithm derivation and transformation, which can be traced back to the so-called *Bird-Meertens formalism* [12] and the foundational work of T. Hagino [15].

Technically, this amounts to the systematic use of the universal property of coinductive extension. I.e., the existence, for each arbitrary coalgebra $\langle U, p : U \longrightarrow \mathcal{P}(Act \times U)\rangle$, of a unique morphism $[\![p]\!]$ to the final coalgebra $\omega : \nu \longrightarrow \mathcal{P}(Act \times \nu)$ satisfying

$$k = [\![p]\!] \;\Leftrightarrow\; \omega \cdot k = \mathcal{P}(\mathsf{id} \times k) \cdot p \tag{1}$$

where $\mathcal{P}$ is the finite powerset functor [5]. Therefore, processes being the inhabitants of the final coalgebra, expression $\mathcal{P}(Act \times \nu)$ stands for a set of pairs each one representing a transition and a corresponding continuation process. Such $[\![p]\!]$ represents the behaviour generated by p and comes equipped with a bunch of laws useful in calculation.

Process combinators are defined either in a direct way (if they are consumed by transitions) or by coinductive extension (if permanent). Examples in the first group are the *inactive* process $\mathbf{0}$, whose set of observations is empty, and non deterministic choice $+$, whose observations are the union of the possible observation upon its arguments [6] Clearly, *prefix* $(\alpha \cdot p)$ is another example. The second group contains all combinators recursively defined. Although this is not the place for a detailed account, we shall briefly review the specification of both *parallel composition* and *synchronous product*, not only because these combinators are used in the paper to join independent web services, but also because they make concrete the notion of *parametrization* by an interaction discipline discussed above. However, to do this, we need first to introduce the *interleaving* combinator.

Interleaving $||| : \nu \times \nu \longrightarrow \nu$ represents an interaction-free form of parallel composition. Observations over the interleaving of two processes correspond to all possible interleavings of observations of their arguments. Thus, $||| = [\![\alpha_{|||}]\!]$, where [7]

$$\begin{aligned}\alpha_{|||} = \nu \times \nu &\xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \xrightarrow{(\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)} (\mathcal{P}(Act \times \nu) \times \nu) \times (\nu \times \mathcal{P}(Act \times \nu)) \\ &\xrightarrow{\tau_r \times \tau_l} \mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu))\end{aligned}$$

Synchronous product models the simultaneous execution of its two arguments. In each step, processes interact through the actions they realize. Let us, for the moment, represent such interaction by a function $\theta : Act \longrightarrow Act \times Act$. Formally, $\otimes = [\![\alpha_{\otimes}]\!]$ where

$$\alpha_{\otimes} = \nu \times \nu \xrightarrow{(\omega \times \omega)} \mathcal{P}(Act \times \nu) \times \mathcal{P}(Act \times \nu) \xrightarrow{\mathsf{sel} \cdot \delta_r} \mathcal{P}(Act \times (\nu \times \nu))$$

[5] The definition generalizes, of course, to an arbitrary coalgebra.

[6] Formally, recalling that final coalgebra ω gives, for each process denotation, the set of its observations, one would write $\omega \cdot \mathbf{0} = \emptyset$ and $\omega \cdot + = \cup \cdot (\omega \times \omega)$, respectively. In a pointwise notation the latter equation becomes $\omega(p + q) = \omega(p) \cup \omega(q)$.

[7] Morphisms $\tau_r : \mathcal{P}(Act \times X) \times C \longrightarrow \mathcal{P}(Act \times (X \times C))$ and $\tau_l : C \times \mathcal{P}(Act \times X) \longrightarrow \mathcal{P}(Act \times (C \times X))$ stand for, respectively, the right and left *strength* associated to functor $\mathcal{P}(Act \times \mathsf{Id})$.

where sel filters out all synchronisation failures (*i.e.*, cases in which $a\theta b = 0$, see below) and δ_r is given by

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

The fundamental point to note is that the definition is parametric on θ, which encodes an *interaction discipline*. Technically, an *interaction discipline* is modeled as an Abelian positive monoid $\langle Act; \theta, 1\rangle$ with a zero element 0. The intuition is that θ determines the interaction discipline whereas 0 represents the absence of interaction: for all $a \in Act$, $a\theta 0 = 0$. On the other hand, being a positive monoid entails $a\theta a' = 1$ iff $a = a' = 1$. A typical example of an interaction structure captures action co-occurrence, in which case θ is defined as $a\theta b = \langle a, b\rangle$, for all $a, b \in Act$. Another example is provided by the action complement match used in CCS [21]. In the sequel we shall introduce a number of specifications for θ suitable to express web service orchestration.

Parallel composition combines the effects of both *interleaving* $|||$ and *synchronous product* $\otimes$. Such a combination is performed at the *genes* level: $| = [\![\alpha_|]\!]$, where

$$\begin{aligned}\alpha_|\ =\ \nu \times \nu \xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) &\xrightarrow{(\alpha_{|||} \times \alpha_\otimes)} \\ \mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) &\xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu))\end{aligned}$$

2.3 Use Patterns and Interaction

Once defined a *parametric* semantics for parallel composition, we may return to the definition of *use patterns* for web services.

Definition 2.2 Let $\mathcal{P}$ be the set of port identifiers and S stand for (the specification of) a web service. Its use pattern, denoted by $use(S)$ is given by a process expression over $Act \triangleq \mathcal{P}\mathcal{P}$, given by the following grammar:

$$\begin{aligned}P\ ::=\ &\mathbf{0}\ |\ \alpha.P\ |\ P + P\ |\ P \otimes P\ |\ P\ |||\ P\ |\ P; P\ |\ P\ |\ P\ | \\ &\sigma\ P\ |\ \mathsf{fix}\ (x\ =\ P)\end{aligned}$$

where α is an element of Act (*i.e.*, a set of port identifiers) and σ is a substitution.

Notice that choosing Act as a *set* of port identifiers allows for the synchronous activation of several ports in a single computational step. The semantics of such expressions is fairly standard, but for the parametrization of all forms of parallel composition (*i.e.*, $\otimes$ and $|$) by an interaction discipline as discussed above. The reader is referred to [27] for the full details. Combinators $\mathbf{0}$, ., +, $|$, $\otimes$ and $|||$, were already introduced in the previous sub-section. Renaming is given by term substitution. The $\mathsf{fix}\ (X\ =\ P)$ is a fixed point construction, which, as usual, can abbreviated in an explicit recursive definition.

Sequential composition, as in Csp [16], is given by ';' and requires its first argument to be a terminating process. Symbol § represents a successfully terminating process, *i.e.*, a process that engages in the success event, ✓, and then stops. Formally, § $\stackrel{\text{abv}}{=}$ ✓.**0**.

The approach proposed in this paper precludes direct interaction between web services — all interaction being mediated by a specific connector. Therefore, if two web services are active in a particular application, their joint behaviour will allow the realization of both use patterns either simultaneously or in an independent way. Formally,

Definition 2.3 The joint behaviour of a collection $\{S_i |\ i \in n\}$ of web services is given by

$$use(S_1) \mid \ldots \mid use(S_n)$$

where the interaction discipline is fixed by $\theta = \cup$, *i.e.*, the synchronisation of actions in α and β corresponds to the simultaneous realization of all of them.

This joint behaviour is computed by the application of Milner's *expansion law* [8] , while obeying to the interaction discipline given by θ. The following example illustrates this construction.

Example 2.4 Consider a service S_1 with two ports a and b whose use pattern is restricted to the activation of either a or b, forbidding their simultaneous occurrence. The expected behaviour is captured by

$$use(S_1) \ = \ \mathsf{fix}\ (x \ = \ a.x \ + \ b.x)$$

Now consider another service, S_2, with ports c and d whose behaviour is given by the co-occurrence of actions in both ports. Therefore,

$$use(S_2) \ = \ \mathsf{fix}\ (x' \ = \ cd.x'), \quad \text{where,}\ \ cd \ \stackrel{\text{abv}}{=} \ \{c, d\}$$

According to definition 2.3, the joint behaviour of S_1 and S_2 is

$$use(S_1) \mid use(S_2) \ = \ \mathsf{fix}\ (x \ = \ acd.x \ + \ bcd.x \ + \ a.x \ + \ b.x \ + \ cd.x)$$

As a final example, consider still another service S_3, with ports e and f activated in strict order, *i.e.*,

$$use(S_3) \ = \ \mathsf{fix}\ (y \ = \ e.f.y)$$

Clearly, expansion leads to

$$use(S_2) \mid use(S_3) \\ = \ \mathsf{fix}\ (x \ = \ cd.x \ + \ e.f.x \ + \ cde.f.x \ + \ cde.cdf.x \ + \ e.cdf \cdots cdf.x)$$

[8] This law, which states that a process is always equivalent to the non deterministic choice of its derivatives, is a fundamental result in interleaving models for concurrency.

3 Configurations

The fundamental notion proposed in this paper as a basis for the orchestration of web services is that of a *configuration*. As explained in the Introduction, this captures the intuition that web services cooperate through specific *connectors* which abstracts the idea of an intermediate *glue code* to handle interaction. Having already defined a notion of web service *interface*, which records all what may be assumed to be known by the web service user, we shall now complete the picture by defining

- what *connectors* are and how they compose;
- the way web services' *interfaces* and *connectors* interact in a configuration.

These points are tackled in the following sub-sections. As one would expect, the two forms of composition (of connectors with themselves and with web services' interfaces) follow *different* interaction disciplines, captured by specific definitions of θ.

3.1 Connectors

Connectors are *glueing devices* between services which ensure the flow of data and the meet of synchronization constraints. Their specification builds on top of our previous work on component interconnection [9], which is extended here with an explicit annotation of activation, or *use*, patterns for their *ports*.

Ports are *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. Another particular characteristic is the ability to construct complex connectors out of simpler ones using a set of *combinators*.

Let $\mathbb{C}$ be a connector with m input and n output ports. Assume, again, $\mathbb{D}$ as a generic type of data values and $\mathbb{P}$ as a set of (unique) *port identifiers* Formally, the behaviour of a connector may be given by

Definition 3.1 The specification of a connector $\mathbb{C}$ is given by a relation $\mathsf{data}.[\![\mathbb{C}]\!] : \mathbb{D}^n \longleftarrow \mathbb{D}^m$ which records the flow of data, and a process expression $\mathsf{port}.[\![\mathbb{C}]\!]$ which gives the pattern of port activation.

Let us illustrate this definition with a number of examples.

3.1.1 Synchronous channel.

The *synchronous channel* has two ports of opposite polarity. This connector forces input and output to become mutually blocking, in the sense that any of them must wait for the other to be completed.

$$\mathsf{data}.[\![\bullet \longmapsto \bullet]\!] = \mathsf{Id}_{\mathbb{D}} \;\; \textit{and} \;\; \mathsf{port}.[\![\bullet \longmapsto \bullet]\!] = \mathsf{fix}\,(x = ab.x)$$

Its semantics is simply the identity relation on data domain $\mathbb{D}$ and its behaviour is captured by the simultaneous activation of its two ports.

3.1.2 Unreliable channel.
Any coreflexive relation, that is any subset of the identity, provides channels which can loose information, thus modelling unreliable communications. Therefore, we define, an *unreliable channel* as

$$\mathsf{data}.[\![\bullet \overset{\diamond}{\longmapsto} \bullet]\!] \subseteq \mathsf{Id}_{\mathbb{D}} \;\; and \;\; \mathsf{port}.[\![\bullet \overset{\diamond}{\longmapsto} \bullet]\!] = \mathsf{fix}\,(x = ab.x + a.x)$$

The behaviour is given by a choice between a successful communication, represented by the simultaneous activation of the ports or, by a failure, represented by the single activation of the input port.

3.1.3 Filter channel.
This is a channel in which some messages are discarded in a controlled way, according to a given predicate $\phi : \mathbf{2} \longleftarrow \mathbb{D}$. Noting that any predicate ϕ can be seen as a relation $\mathsf{R}_\phi : \mathbb{D} \longleftarrow \mathbb{D}$ such that $d\mathsf{R}_\phi d'$ iff $d = d' \wedge (\phi\, d)$, define

$$\mathsf{data}.[\![\bullet \overset{\phi}{\longmapsto} \bullet]\!] = \mathsf{R}_\phi \;\; and \;\; \mathsf{port}.[\![\bullet \overset{\phi}{\longmapsto} \bullet]\!] = \mathsf{fix}\,(x = ab.x)$$

3.1.4 Drain.
A drain has two input, but no output, ports. Therefore, it looses any data item crossing its boundaries. A drain is *synchronous* if both write operations are requested to succeed at the same time (which implies that each write attempt remains pending until another write occurs in the other end-point). It is *asynchronous* if, on the other hand, write operations in the two ports do not coincide. The formal definitions are, respectively,

$$\mathsf{data}.[\![\bullet \overset{\blacktriangledown}{\longmapsto\!\!\dashv} \bullet]\!] = \mathbb{D} \times \mathbb{D} \;\; and \;\; \mathsf{port}.[\![\bullet \overset{\blacktriangledown}{\longmapsto\!\!\dashv} \bullet]\!] = \mathsf{fix}\,(x = ab.x)$$

and,

$$\mathsf{data}.[\![\bullet \overset{\triangledown}{\longmapsto\!\!\dashv} \bullet]\!] = \mathbb{D} \times \mathbb{D} \;\; and \;\; \mathsf{port}.[\![\bullet \overset{\triangledown}{\longmapsto\!\!\dashv} \bullet]\!] = \mathsf{fix}\,(x = a.x + b.x)$$

3.1.5 $Fifo_1$.
This is a channel with a buffer of a single position.

$$\mathsf{data}.[\![\bullet \longmapsto \bullet]\!] = \mathsf{Id}_{\mathbb{D}} \;\; and \;\; \mathsf{port}.[\![\bullet \longmapsto \bullet]\!] = \mathsf{fix}\,(x = a.b.x)$$

3.2 Combining Software Connectors

Connectors can be combined to build more complex *glueing code*. The following are the required combinators.

3.2.1 Aggregation.
This combinator places its arguments side-by-side, with no direct interaction between them.

$$\mathsf{port}.[\![\mathbb{C}_1 \boxtimes \mathbb{C}_2]\!] = \mathsf{port}.[\![\mathbb{C}_1]\!] \mid \mathsf{port}.[\![\mathbb{C}_2]\!] \tag{2}$$

with $\theta = \cup$.

3.2.2 Hook.
This combinator encodes a *feedback* mechanism, drawing a direct connection between an output and an input port. Formally, $\mathsf{port}.[\![\mathbb{C} \curvearrowleft_i^j]\!]$ is obtained from $\mathsf{port}.[\![\mathbb{C}]\!]$, by deleting references to ports i and j. To be well-formed it is required that i and j appear in different factors of some form of parallel composition ($|||$, $\otimes$, or $|$).

Its effect on data is modelled by the following relation:

$$\begin{array}{l} R : \mathbb{D}^m \longleftarrow \mathbb{D}^n \\ R \curvearrowleft_i^j : \mathbb{D}^{m-1} \longleftarrow \mathbb{D}^{n-1} \\ t = t_m, \ldots, t_{i+i}, t_{i-i}, \ldots, t_0, \text{ and } t' = t'_n, \ldots, t'_{j+i}, t'_{j-i}, t'_0 \\ t(R \curvearrowleft_i^j)t' \text{ iff } \exists_x.(t_n, \ldots, t_{i+i}, x, t_{i-i}, \ldots, t_0)R(t_m, \ldots, t_{j+i}, x, t_{j-i}, \ldots, t_0) \end{array}$$

3.2.3 Join.
Its effect is to plug ports with same polarity. The aggregation of input ports is done by a *right join* ($\mathbb{C} \overset{i}{j} > z$), where $\mathbb{C}$ is a connector, and 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 in a nondeterministic way. On the other hand, aggregation of *output ports* resorts to a *left join* ($z <_j^i \mathbb{C}$). This behaves like a *broadcaster* sending synchronously messages from z to both i and j. Formally, at a behavioural level, both operators effect is that of a renaming operation

$$\mathsf{port}.[\![(\mathbb{C} \overset{i}{j} > n)]\!] = \mathsf{port}.[\![(n <_j^i \mathbb{C})]\!] = \{n \leftarrow i, n \leftarrow j\}\mathsf{port}.[\![\mathbb{C}]\!]$$

The differences just mentioned are specified at the data level by,
Left join:

$$\begin{array}{l} R : \mathbb{D}^m \longleftarrow \mathbb{D}^n \\ (R \overset{i}{j} > z) : \mathbb{D}^{m-1} \longleftarrow \mathbb{D}^n \\ t = t_m, \ldots, t_{i+i}, t_{i-i}, \ldots, t_{j+i}, t_{j-i}, \ldots, t_0, \text{ and} \\ t' = t'_n, \ldots, t_{i+i}, t_{i-i}, t'_{j+i}, t'_{j-i}, \ldots, t'_0 \\ t(R \overset{i}{j} > z)t' \text{ iff} \\ \quad \exists_x.(t_z, t_n, \ldots, t_{i+i}, t_z, t_{i-i}, \ldots, t_{j+i}, x, t_{j-i}, \ldots, t_0)Rt' \end{array}$$

Right join:

$$
\begin{array}{l}
R : \mathbb{D}^m \longleftarrow \mathbb{D}^n \\
(z <^i_j R) : \mathbb{D}^m \longleftarrow \mathbb{D}^{n-1} \\
t = t_m, \ldots, t_{i+i}, t_{i-i}, \ldots, t_{j+i}, t_{j-i}, \ldots, t_0, \text{ and} \\
t' = t'_n, \ldots, t_{i+i}, t_{i-i}, t'_{j+i}, t'_{j-i}, \ldots, t'_0 \\
t(z <^i_j R)t' \text{ iff} \\
\quad \exists_x . tR(t_z, t_m, \ldots, t_{j+i}, t_{j-i}, \ldots, t_{i+i}, t_{i-i}, \ldots, t_0)
\end{array}
$$

3.3 Configurations

A *configuration* is simply a collection of web services, characterized by their interfaces, interconnected through an *orchestrator*, *i.e.*, a connector network built from elementary connectors using the combinators mentioned above. Formally,

Definition 3.2 A configuration involving a collection $S = \{S_i |\ i \in n\}$ of web services is a tuple

$$\langle U, \mathbb{C}, \sigma \rangle \tag{3}$$

where $U = use(S_1) \mid use(S_2) \mid \cdots \mid use(S_n)$ is the (joint) use pattern for S, $\mathbb{C}$ is a connector and σ a mapping of ports in S to ports in $\mathbb{C}$.

The relevant point concerning configurations is the semantics of the interaction between the *connector's port behaviour* and the *joint use patterns* of the involved web services. This is captured by a synchronous product $\otimes$ for a quite peculiar θ, which is expected to capture the following requirements:

- Interaction is achieved by the simultaneous activation of identically named ports [9] .
- There is no interaction if the connector intends to activate ports which are not linked to the ones offered by the web services' side. For example if a port a of a service S is connected to the input end of a synchronous channel whose output end is disconnected, no information can flow and port a will never be activated.
- The dual situation is allowed, *i.e.*, if the web services' side offer activation of all ports plugged to the ones offered by the connectors' side, their intersection is the resulting interaction.
- Moreover, and finally, activation of unplugged web services' ports is always possible.

Formally, this is captured in the following definition.

[9] Often this will force the introduction of suitable port renamings.

Definition 3.3 The behaviour $bh(\Gamma)$ of a configuration $\Gamma = \langle U, \mathbb{C}, \sigma \rangle$ is given by

$$bh(\Gamma) \;=\; \sigma\, U \;\otimes\; \mathsf{port}.[\![\mathbb{C}]\!] \tag{4}$$

where θ underlying the $\otimes$ connective is given by

$$c\;\theta\;c' \;=\; \begin{cases} c \cap (c' \cup \mathsf{free}) & \Leftarrow\; c' \subseteq c \\ \emptyset & \Leftarrow\; otherwise \end{cases} \tag{5}$$

and free denotes the set of unplugged ports in U, *i.e.*, not in the domain of mapping σ.

In the sequel the use of configurations, and the computation of their behaviours, is illustrated by two examples.

3.4 Examples

Example 3.4 Our first example is taken from [14]. Suppose an organization offers a "Holiday Reservation Service" (HRS) that allows customers to organize holiday travels. A possible configuration HR is given by

$$HR \;=\; \langle WHR,\, SB,\, \sigma_{HS} \rangle$$

where

$$\begin{aligned} WHR \;&= use(HRS) \mid use(HORS) \mid use(FRS) \mid use(CRS) \\ \sigma_{HS} \;&= \{a \leftarrow A, b \leftarrow B, c \leftarrow C, d \leftarrow D, e \leftarrow E, f \leftarrow F, g \leftarrow G\} \end{aligned}$$

It is depicted in Fig 1.

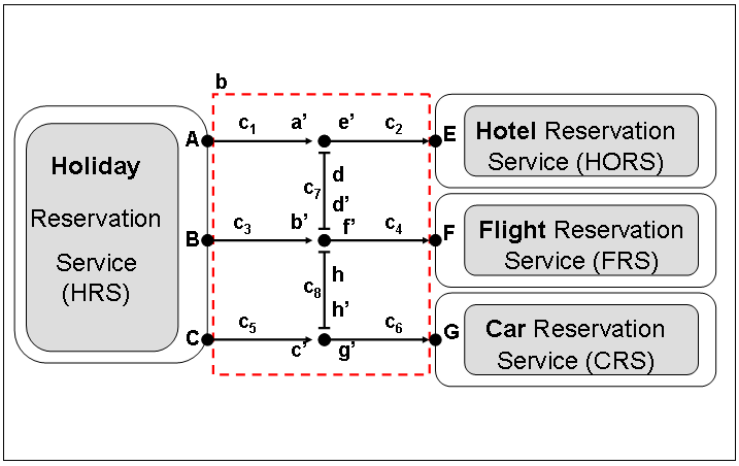

Fig. 1. Holiday Reservation

It is assumed that organizing holiday requires making reservations for a hotel, for a flight and for a car. Some other organizations offer services to deal

with each part of the job: hence the HORS, FRS and CRS services. Before asking the customer to pay, the HRS services needs to *commit* a transaction containing each of the reservations. A holiday reservation should only succeed when all other three reservations succeed.

The *commit* requirement is modeled by a particular external *glue code*: a *barrier synchronization* connector consisting of six synchronous channels and two synchronous drain channels, organized together as in Fig. 1.

Let us now compute the overall behaviour of configuration HR. After port renaming, the usage pattern of each web service is as follows:

$$\begin{aligned} use(HRS) &= \mathsf{fix}\ (x = a.x + b.x + c.x + abc.x) \\ use(HORS) &= \mathsf{fix}\ (x = e.x) \\ use(FRS) &= \mathsf{fix}\ (x = f.x) \\ use(CRS) &= \mathsf{fix}\ (x = g.x) \end{aligned}$$

Its joint behaviour is given their $|$-composition, with $\theta = \cup$.

On the other hand, the behaviour of connector SB is obtained by the composition of the behaviours of the six elementary connectors aggregated through the combinators:

$$\mathsf{port}.[\![SB]\!] = \mathsf{fix}\ x = abcefg.x \tag{6}$$

This is computed starting from the behaviours of the elementary connectors

$$\begin{aligned} &\mathsf{port}.[\![c_1]\!] = \mathsf{fix}\ (x = aa'.x),\ \mathsf{port}.[\![c_2]\!] = \mathsf{fix}\ (x = e'e.x), \\ &\mathsf{port}.[\![c_3]\!] = \mathsf{fix}\ (x = bb'.x),\ \mathsf{port}.[\![c_4]\!] = \mathsf{fix}\ (x = f'f.x), \\ &\mathsf{port}.[\![c_5]\!] = \mathsf{fix}\ (x = cc'.x),\ \mathsf{port}.[\![c_6]\!] = \mathsf{fix}\ (x = g'g.x), \\ &\mathsf{port}.[\![c_7]\!] = \mathsf{fix}\ (x = dd'.x),\ \mathsf{port}.[\![c_8]\!] = \mathsf{fix}\ (x = hh'.x) \end{aligned}$$

as follows

$$\begin{aligned} Cn_1 &= \mathsf{port}.[\![(n <_d^{e'} (c_2 \boxtimes c_7))]\!] = \mathsf{fix}\ (x = \{n \leftarrow e', n \leftarrow d\}\ ee'dd'.x) \\ Cn_2 &= \mathsf{port}.[\![((c_1 \boxtimes Cn_1)\ \curlywedge_{a'}^{n})]\!] = \mathsf{fix}\ (x = aed'.x) \\ Cn_3 &= \mathsf{port}.[\![(m <_{h'}^{g'} (c_6 \boxtimes c_8))]\!] = \mathsf{fix}\ (x = \{m \leftarrow g', m \leftarrow h'\}\ hh'gg'.x) \\ Cn_4 &= \mathsf{port}.[\![((Cn_3 \boxtimes c_5)\ \curlywedge_{c'}^{m})]\!] = \mathsf{fix}\ (x = cgh.x) \\ Cn_5 &= \mathsf{port}.[\![(z <_{f'}^{d'} (Cn_2 \boxtimes c_4))]\!] = \mathsf{fix}\ (x = \{z \leftarrow d', z \leftarrow f'\}\ add'ff'.x) \\ Cn_6 &= \mathsf{port}.[\![((Cn_5 \boxtimes c_3)\ \curlywedge_{b'}^{z})]\!] = \mathsf{fix}\ (x = abef.x) \\ \mathsf{port}.[\![SB]\!] &= \mathsf{port}.[\![((Cn_6 \boxtimes Cn_4)\ \curlywedge_{h}^{z})]\!] = \mathsf{fix}\ (x = abcefg.x) \end{aligned}$$

The result of the $\otimes$ composition of WHR and (6) is the behaviour of configuration HR. There is no need, however, to compute the complete expansion

of the parallel composition in WHR expression, which is

$$\begin{aligned}\mathsf{fix}\,(x \;= & a.x + \cdots + e.x + f.x + g.x + \\ & ae.x + \cdots + be.x + \cdots + ce.x + \cdots + abce.x + \cdots + \\ & aef.x + \cdots + bef.x + \cdots + cef.x + \cdots + abcef.x + \cdots + \\ & aefg.x + \cdots + befg.x + \cdots + cefg.x + \cdots + \underline{abcefg}.x + \cdots + \\ & ef.x + eg.x + fg.x + efg.x)\end{aligned}$$

because, according to interaction discipline (5), the only successful case of composition with $\mathsf{port}.[\![SB]\!]$ corresponds to the underlined alternative in the expression above. Clearly, the θ-composition of $abcefg$ with $abcefg$ (from the connector side) is $abcefg$, while for all other cases it results in the empty set $\emptyset$. Therefore, and finally,

$$bh(HR) \;=\; \mathsf{fix}\,(x \;=\; abcefg.x) \tag{7}$$

Example 3.5 As a second example consider an elementary *banking system* composed by an ATM machine, a $Bank$, and a $DBRep$ service whose purpose is to backup all the messages flowing through the connector. Therefore, all messages are replicated before being stored. Configuration BS, depicted in Fig. 2, is specified as

$$BS \;=\; \langle WBS,\, DBC,\, \sigma_{BS} \rangle$$

where

$$\begin{aligned} WBS \;&= use(ATM) \mid use(Bank) \mid use(DBRep) \\ \sigma_{HS} \;&= \{a \leftarrow A_{rq}, e \leftarrow A_{rs}, c \leftarrow DB_r, f \leftarrow DB_p, d \leftarrow B_{rs}, b \leftarrow B_{rq}\} \end{aligned}$$

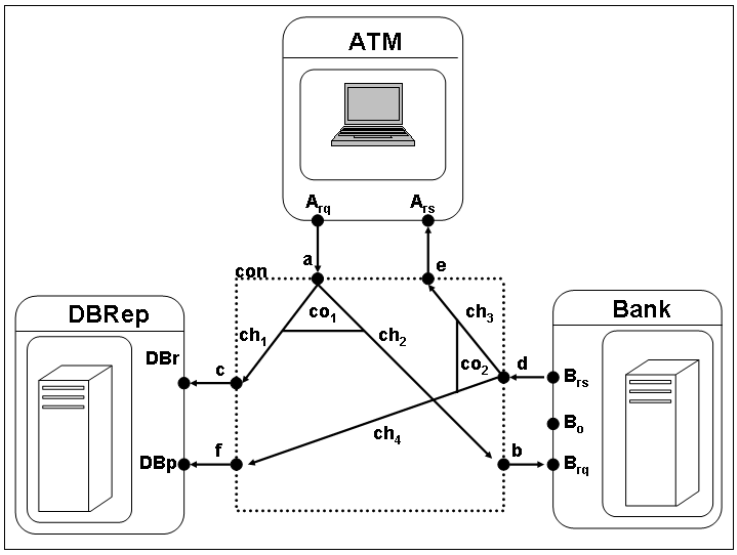

Fig. 2. Bank System

The use patterns of each web service are as follows

$$\begin{aligned} use(ATM) &= \mathsf{fix}\ (x = a.e.x) \\ use(Bank) &= \mathsf{fix}\ (y = b.d.y) \\ use(DBRep) &= \mathsf{fix}\ (z = c.z + f.z) \end{aligned}$$

Connector DBC behaves like a double broadcaster (hence its name). Its behaviour allows for the both simultaneous or independent activation of each broadcast (co_1 or co_2) as shown by the following computation:

$$\begin{aligned} &\mathsf{port}.[\![ch_1]\!] = \mathsf{fix}\ (x = b'b.x),\ \mathsf{port}.[\![ch_2]\!] = \mathsf{fix}\ (x = c'c.x) \\ &\mathsf{port}.[\![co_1]\!] = \mathsf{port}.[\![(a <^{b'}_{c'} (ch_1 \boxtimes ch_2))]\!] = \mathsf{fix}\ (x = abc.x) \\ &\mathsf{port}.[\![ch_3]\!] = \mathsf{fix}\ (x = e'e.x),\ \mathsf{port}.[\![ch_4]\!] = \mathsf{fix}\ (x = f'f.x) \\ &\mathsf{port}.[\![co_2]\!] = \mathsf{port}.[\![(d <^{e'}_{f'} (ch_3 \boxtimes ch_4))]\!] = \mathsf{fix}\ (x = def.x) \\ &\mathsf{port}.[\![DBC]\!] = \mathsf{port}.[\![(co_1 \boxtimes co_2))]\!] = \mathsf{fix}\ (x = abc.x + def.x + abcdef.x) \end{aligned}$$

Again, to determine $bh(BS)$ one needs to expand WBS, eventually resorting to some tool support [10], as the number of alternatives is rather high. According to (5), we shall only look at sets of ports in prefix expressions which contain port set prefix in $\mathsf{port}.[\![DBC]\!]$ expression. For the first level of expansion alternative $abc.(e.x \mid d.y \mid z)$ is the only one to θ-compose with abc in $\mathsf{port}.[\![DBC]\!]$, resulting in abc again. Then, consider the expansion of term $(e.x \,|\, d.y \,|\, z)$: the only alternative to worth consider (*i.e.*, which does not lead to $\emptyset$ on θ-composition) is $edf.(x \mid y \mid z)$, the resulting interaction being edf. From this point on the same expansion pattern repeats. This means that $bh(BS)$ becomes:

$$bh(BS) = \mathsf{fix}\ (x = abc.edf.x) \tag{8}$$

Notice how the particular use patterns in the web services act as a *constraint* over the admissible behaviour of connector DBC.

This example may be also used to check how definition (5) deals with the presence of unplugged ports, such us port B_o in service $Bank$. Consider, then, the following two alternatives for the use pattern of service $Bank$:

$$use(Bank) = \mathsf{fix}\ (y = bB_o.d.y) \tag{9}$$

$$use(Bank) = \mathsf{fix}\ (y = b.d.y + B_o.y) \tag{10}$$

In the expansion of WBS, case (9), which captures the simultaneous activation of ports b and B_o, leads to term $abB_oc.(e.x \,|\, d.y \,|\, z)$ which, as $\mathsf{free} = \{B_o\}$ leads to

$$bh(BS) = \mathsf{fix}\ (x = abcB_o.edf.x) \tag{11}$$

[10] A handy alternative is the Cwb tool [22].

Case (10) specifies that ports b and B_o are activated in alternative: no term with both b and B_o will appear in the expansion and, therefore, $bh(BS)$ remains as given by equation (8).

4 Conclusions and Future Work

Service-oriented computing is an emerging paradigm for distributed computing with increasing impact on the way modern software systems are designed and developed. Services are autonomous and heterogeneous computational entities which cooperate, following a loose coupling discipline, to achieve some complex goals. Web services are one of the most prominent technologies in this paradigm. As an emerging technology, however, it still lacks not only sound semantical *models* but also suitable *calculi* to reason about and transform service-oriented designs.

Having proposed formal models for both *behavioural interfaces* and *configurations*, as a base for representing web services' orchestration, this paper may be a step in that direction. The approach combines two ingredients in which the authors have been working for some time now: *exogenous coordination* and a methodology for the *design of process algebras parametric on the interaction discipline*.

Lots of questions, however, remain open. Let us enumerate the ones in which we are currently involved.

Negative Information. In a number of practical situations service orchestration also depends on what may be called *negative information*. One of the basic channels considered in Reo [5] is the *lossy channel* which acts as a synchronous one if both an input and an output request are pending on its two ports, but will loose the data item on the input on the absence of an output request on the other port. Notice this behaviour is distinct from that of the *unreliable* channel discussed above, which looses data non deterministically.

To handle these cases we enrich the specification of *Act* in definition 2.2 to include *negative port activations*, or more rigorously stated, *absence of port requests*, denoted, for each port p, by $\tilde{p}$. Therefore, the specification of Reo's lossy channel becomes possible as

$$\mathsf{data}.[\![\bullet \overset{\cdots}{\longmapsto} \bullet]\!] \subseteq \mathsf{Id}_{\mathbb{D}} \;\; \textit{and} \;\; \mathsf{port}.[\![\bullet \overset{\cdots}{\longmapsto} \bullet]\!] = \mathsf{fix}\,(x = ab.x + a\tilde{b}.x)$$

The following step is to modify θ in (5), definition 3.3, so that

$$c\;\theta\;c' = 0 \quad \Leftarrow \quad \exists_{p \in \mathcal{P}}.\; p \in c \;\wedge\; \tilde{p} \in c' \tag{12}$$

Then the joint behaviour of a configuration orchestrated by a lossy channel and involving *use* pattern $U = \mathsf{fix}\,(x = ab.x)$ is computed as follows

$$U \otimes \mathsf{port}.[\![\bullet \overset{\cdots}{\longmapsto} \bullet]\!] = \mathsf{fix}\,(x = (ab\theta ab).x + (ab\theta a\tilde{b}).x) = \mathsf{fix}\,(x = ab.x)$$

whereas

$$U \otimes \mathsf{port}.[\![\bullet \overset{\cdots}{\longmapsto} \bullet]\!] = \mathsf{fix}\,(x = (a\theta ab).x + (a\theta a\tilde{b}).x) = \mathsf{fix}\,(x = a.x + a.x)$$
$$= \mathsf{fix}\,(x = a.x)$$

for $U\mathsf{fix}\,(x = a.x)$, *i.e.*, in the absence of an output request from one of the services.

Workflow Patterns. The notion of behavioural interface discussed in section 2 is close to that of *workflow patterns* [1] whose role in the design of service-oriented systems is well recognized. Their formalization is still an 'hot' research topic (see, *e.g.*, [2,30,4], among many others). We believe all such patterns can be encoded in our notion of behavioural interface, provided the latter is enriched with two port attributes to keep track of the number of port requests and activations. Fig. 3 illustrates the specification of two common patterns, part of a systematic classification effort currently under development.

Pattern 2 (Parallel split)
Description. A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.
Specification.

$$use(\mathsf{WS}_2) = P_1 \mid \ldots \mid P_n$$

Pattern 3 (Synchronization)
Description. A point in the workflow process where multiple parallel activities converge into one single thread of control, thus synchronizing multiple threads.
Specification.

$$use(\mathsf{WS}_3) = (a_1.a_2.\ldots.a_n.\S \otimes b_1.b_2.\ldots.b_n.\S)\,;\ P$$

Fig. 3. Two workflow patterns and their encoding as use patterns

For the general case, however, this requires not only the inclusion in the just mentioned interface of port attributes, but also a conditional constructor $(c \rightarrow P, P')$, where c is a boolean condition on those attributes.

Mobility. It is not clear how the model discussed in this paper can be extended to cope with mobility issues, and, in particular, with dynamic reconfiguration of web services networks. The question is, in fact, more general:

we still know very little about the semantics of mobility in the context of exogenous coordination. Tentative solutions in *e.g.*, Reo [6] or our own contribution documented in [11], are still of an operational nature.

Language Semantics. Whether formal models, like the one discussed in this paper, can be of use in providing precise semantic foundations of emerging languages for web services composition (*e.g.*, `BPEL4WS`, `XLANG` or `WS-CDL`, among others) remains a challenge we intend to face in future work. Its relevance, from the point of view of software engineering, cannot be underestimated.

Acknowledgement

This research was carried on in the context of the PURe Project supported by Fct, the Portuguese Foundation for Science and Technology, under contract `POSI/ICHS/44304/2002`.

References

[1] W. M. P. V. D. Aalst, A. H. M. T. Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[2] N. R. Adam, V. Atluri, and W.-K. Huang. Modeling and analysis of workflows using petri nets. *J. Intell. Inf. Syst.*, 10(2):131–158, 1998.

[3] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM TOSEM*, 6(3):213–249, 1997.

[4] R. Amici, F. Corradini, and E. Merelli. A process algebra view of coordination models with a case study in computational system biology. In L. Bocchi and P. Ciancarini, editors, *Proceedings of the First International Workshop on Petri Nets and Coordination (PNC04), Satellite Event of the 25th International Conference on Application and Theory of Petri Nets, Bologna, Italy, June 21, 2004*, pages 33–47, 2004.

[5] F. Arbab. Reo: a channel–based coordination model for component composition. *Mathematical Structures in Comp. Sci.*, 14(3):329–366, 2004.

[6] F. Arbab and F. Mavadatt. Coordination through channel composition. In *Proc. Coordination Languages and Models*. Springer Lect. Notes Comp. Sci. (2315), 2002.

[7] 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.

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

[9] M. Barbosa and L. Barbosa. Specifying software connectors. In K. Araki and Z. Liu, editors, *Proc. First International Colloquim on Theoretical Aspects of Computing (ICTAC'04), Guiyang, China*, pages 53–68. Springer Lect. Notes Comp. Sci. (3407), 2004.

[10] M. A. Barbosa and L. S. Barbosa. A relational model for component interconnection. In *Journal of Universal Computer Science*, volume 10, pages 808–823, 2004.

[11] M. A. Barbosa and L. S. Barbosa. An orchestrator for dynamic interconnection of software components. In *Proc. 2nd International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord'06)*, Bologna, Italy, June 2006. Elsevier.

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

[13] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing web service choreographies. *Electr. Notes Theor. Comput. Sci.*, 105:73–94, 2004.

[14] N. K. Diakov and F. Arbab. Compositional construction of web services using reo. In *WSMAI*, pages 49–58, 2004.

[15] T. Hagino. A typed lambda calculus with categorical type constructors. In D. H. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Category Theory and Computer Science*, pages 140–157. Springer Lect. Notes Comp. Sci. (283), 1987.

[16] C. A. R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice-Hall International, 1985.

[17] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–159, 1997.

[18] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, London, UK, 1995. Springer-Verlag.

[19] J. Magee, J. Kramer, and D. Giannakopoulou. Behaviour analysis of software architectures. In *WICSA1: Proc. of the TC2 First Working IFIP Conf. on Software Architecture (WICSA1)*, pages 35–50. Kluwer, B.V., 1999.

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

[21] R. Milner. *Communicating and Mobile Processes: the π-Calculus.* Cambridge University Press, 1999.

[22] F. Moller and P. Stevens. The edinburgh concurrency workbench (version 7). User's manual, LFCS, Edinburgh University, 1996.

[23] O. Nierstrasz and F. Achermann. A calculus for modeling software components. In F. S. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Proc. First International Symposium on Formal Methods for Components and Objects (FMCO'02)*, pages 339–360. Springer Lect. Notes Comp. Sci. (2852), 2003.

[24] G. Papadopoulos and F. Arbab. Coordination models and languages. In *Advances in Computers — The Engineering of Large Systems*, volume 46, pages 329–400. 1998.

[25] F. Plasil and D. Mikusik. Inheriting synchronization protocols via sound enrichment rules. In *JMLC '97: Proc. of the Joint Modular Languages Conference on Modular Programming Languages*, pages 267–281, London, UK, 1997. Springer-Verlag.

[26] F. Plasil and S. Visnovsky. Behavior protocols for software components. *IEEE Trans. Softw. Eng.*, 28(11):1056–1076, 2002.

[27] P. R. Ribeiro, M. A. Barbosa, and L. S. Barbosa. Generic process algebra: A programming challenge. In *Proc. 10th Brazilian Symposium on Programming Languages*, Itatiaia, Brasil, 2006.

[28] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *ICWS '04: Proc. of IEEE Inter. Conf. on Web Services (ICWS'04)*, page 43, Washington, DC, USA, 2004. IEEE Computer Society.

[29] P. Selinger. Categorical structure of asynchrony. In *MFPS'98 (invited talk), New Orleans*. ENTCS, volume 20, Elsevier, March 1999.

[30] H. Smith and P. Fingar. Workflow is just a pi process, 2003.

[31] S. Visnovsky. *Modeling Software Components Using Behavior Protocols*. Doctoral thesis, Charles University, Czech Replubic, 2003.