Towards a Unifying Theory for Web Services Composition*

Manuel Mazzara[1] and Ivan Lanese[2]

[1] Faculty of Computer Science
Free University of Bozen-Bolzano, Italy
manuel.mazzara@unibz.it
[2] Computer Science Department
University of Bologna, Italy
lanese@cs.unibo.it

Abstract. Recently the term *orchestration* has been introduced to address composition and coordination of *web services*. Several languages used to describe business processes using this approach have been presented, and most of them use the concepts of *long-running transactions* and *compensations* to cope with error handling. WS-BPEL, which is currently the most used orchestration language, also provides a Recovery Framework. However its complexity hinders rigorous treatment. In this paper, we address the notion of orchestration from a formal point of view with particular attention to transactions and compensations. In particular, we introduce $\texttt{web}\pi_\infty$, an untimed version of $\texttt{web}\pi$, and the related theory, as a foundational unifying framework for orchestration able to meet composition requirements and to encode the whole BPEL itself.

1 Web Services

Service Oriented Computing (SOC) [7] is an emerging paradigm for distributed computing and e-business processing that finds its origin in object-oriented and component computing [17]. One of the main goals of SOC is enabling developers to build networks of integrated and collaborative applications, regardless of both the platform where the applications or services run (e.g., the operating system) and the programming language used to develop them.

Web services are a set of technologies supporting SOC. They provide a platform on which applications can be developed by taking advantage of the Internet infrastructure. A web service makes its functionalities available over the network through specific access points, in such a way that they can be exploited, in turn, by other services. Web services are an evolutionary technology, they did not just exist suddenly. There is no revolution about them, this technology has to be seen as an evolution based on the already existing Internet protocols.

1.1 Web Services Composition

The interesting thing in the web services programming model is that a service can itself use several other services and all of them are based on the same model.

* Research partially supported by the Project FET-GC II IST-2005-16004 Sensoria.

M. Bravetti, M. Nuñes, and G. Zavattaro (Eds.): WS-FM 2006, LNCS 4184, pp. 257–272, 2006.

This means that a composite business process can itself be exposed as a web service, enabling business processes to be aggregated to form higher-level processes. There is, indeed, a recursive use of the model where, notably, the overall scenario will be transparent to the final consumer. In this way, web services technologies provide a mechanism to build complex services out of simpler ones: this practice is called *web services composition*. A composition consists in the aggregation of services by programming the relative interactions and has the ability to make the created aggregations reusable [5]. To program a complex cross-enterprise task or a business transaction, for example, it is possible to logically chain discrete web service activities into inter-enterprise business processes.

Different organizations are presently working on additional layers which have to deal with the new approach of composing web services on a workflow base for business automation purposes. Two examples of past proposals for describing service compositions are IBM's WSFL (Web Services Flow Language) [9] and Microsoft's XLANG [18]. XLANG is a block-structured language with basic control flow structures such as `sequence`, `switch` (conditional), `while` (looping), `all` (parallel) and `pick` (choice based on timing or external events). Unlike XLANG, WSFL is not limited to block structures and it allows for arbitrary directed acyclic graphs. Iteration is only supported through exit conditions - that is, an activity iterates until its exit condition is met.

A more recent proposal (presently a working draft by OASIS), which aims at integrating WSFL and XLANG, is the Web Services Business Process Execution Language [2] (WS-BPEL or BPEL for short). It combines WSFL's graph-oriented process representation and XLANG's structural construct-based processes into a unified language for composition. However, while the graph based model used in WSFL has largely not evolved, block-structured programming, similar to the method of describing workflow in XLANG, has evolved incredibly in the last decades to encapsulate complexity and allow for greater manageability and maintainability. Some of the lessons learned from programming could improve business modeling using workflow. The use of block-structured programming can be cited as one of the main points in favor of the approach taken by XLANG and the BizTalk Orchestration framework [12], and in this paper we will focus on it.

Business process orchestration has to meet several requirements, including a way to address concurrency and asynchronous message passing, which form the basic paradigm of the distributed computation on the Internet. Another relevant aspect is the management of exceptions and transactional integrity [15]. BPEL covers all these aspects, but its current specification is rather involved. As far as error handling is concerned, for instance, it provides three different mechanisms for coping with abnormal situations: *fault handling*, *compensation handling* and *event handling*[1]. Documentation is informal and in many points it is not very clear, in particular when interactions among the different mechanisms are required. Therefore the language is difficult to use, and it is relevant to

[1] The BPEL event handling mechanism was not designed for error handling only. However, it can be used for this purpose and we concentrate here on this aspect.

address the issue of error recovering in a formal way to clarify all the controversial aspects.

In order to formally deal with the requirements, we start from the π-calculus [14,16] because the definition of XLANG (and then BPEL) has been strongly influenced by it. Unfortunately, the original π-calculus does not provide any transactional mechanism. For this reason, we consider an extension of the calculus called $\texttt{web}\pi$ [8], which extends the basic calculus with transactional facilities. In particular, we will present an untimed variant (while one of the main concerns of $\texttt{web}\pi$ is time) of it, that we call $\texttt{web}\pi_\infty$, and we analyze its semantic properties. We concentrate, in particular, on the weak behavioral equivalence, which abstracts from internal steps, and which is not meaningful in the $\texttt{web}\pi$ scenario, since time allows to find out internal steps anyway. In fact, internal steps make time to progress, and timeouts to trigger. The most common formalization of behavioral equivalence is through *barbed congruence*, which guarantees that equated processes are indistinguishable by external observers, even when put in arbitrary contexts. For instance equivalent web services remain indistinguishable also when composed to form complex business transactions. As main contributions we show that barbed congruence can be characterized via a labeled semantics that is easier to compute, and we show some examples on how this framework can be used to prove interesting properties about compensations and web services composition. The first author exploited $\texttt{web}\pi_\infty$ to formalize a simplification of the BPEL Recovery Framework unifying all the mechanisms (fault, compensation and event handling), as can be found in [10]. Thus the results therein can be used to derive also properties of BPEL. Further results in this sense and all the complete proofs just sketched in the paper can be found in the Ph.D. thesis of the first author [11].

2 The Orchestration Calculus $\texttt{web}\pi_\infty$

In this section we present $\texttt{web}\pi_\infty$, introducing its syntax and both a reduction semantics and a weak barbed congruence. Notably, $\texttt{web}\pi_\infty$ semantics is not just a simplification of $\texttt{web}\pi$ semantics, since, in the last one, time is used also for transaction commit, while here we have to deal with it differently. Also, we add input-guarded choice to the calculus, which was not present in [8].

The syntax of $\texttt{web}\pi_\infty$ *processes* relies on a countable set of channel *names*, ranged over by $x, y, z, u, \ldots$. Tuples of names (possibly empty) are written $\widetilde{u}$, and $|\widetilde{u}|$ is the length of tuple $\widetilde{u}$. When we write $i \in I$ we intend, if nothing is said, that I is a finite non-empty set of indexes.

$$
\begin{array}{lll}
P & ::= \mathbf{0} & (\textbf{nil}) \\
& \mid \overline{x}\,\widetilde{u} & (\textbf{output}) \\
& \mid \sum_{i \in I} x_i(\widetilde{u_i}).P_i & (\textbf{guarded choice}) \\
& \mid (x)P & (\textbf{restriction}) \\
& \mid P \mid P & (\textbf{parallel composition}) \\
& \mid\, !x(\widetilde{u}).P & (\textbf{guarded replication}) \\
& \mid \langle\!|P \,;\, P|\!\rangle_x & (\textbf{workunit})
\end{array}
$$

A process can be the inert process $\mathbf{0}$, an output $\overline{x}\,\widetilde{u}$ sent on a name x that carries a tuple of names $\widetilde{u}$ (if $\widetilde{u}$ is empty we may write simply $\overline{x}$), a choice among input-guarded processes that consumes a message $\overline{x_i}\,\widetilde{w_i}$ and then behaves like $P_i\{\widetilde{w_i}/\widetilde{u_i}\}$, a restriction $(x)P$ that behaves as P except that inputs and messages on x are prohibited, a parallel composition of processes, a replicated input $!x(\widetilde{u}).P$ that consumes a message $\overline{x}\,\widetilde{w}$ and then behaves like $P\{\widetilde{w}/\widetilde{u}\} \mid !x(\widetilde{u}).P$, or a workunit (or simply a unit) $\langle\!|P\ ;\ Q|\!\rangle_x$ that behaves as the *body* P until an abort $\overline{x}$ is signaled (either by P or from the outside) and then behaves as the *event handler* Q.

We avoid to mix replication and choice since this simplifies the presentation and since this is not necessary for our aims (and notably to model BPEL semantics). The extension is however easy.

We use $+$ to denote binary choice. We use $\prod_{i\in I} P_i$ to denote the parallel composition of processes P_i for each $i \in I$. Names x in outputs, inputs, and replicated inputs are called *subjects*. It is worth to notice that the syntax of $\mathtt{web}\pi_\infty$ processes essentially adds the workunit construct to the asynchronous π-calculus.

The input $x(\widetilde{u}).P$, restriction $(x)P$ and replicated input $!x(\widetilde{u}).P$ are binders of names $\widetilde{u}$, x and $\widetilde{u}$ respectively. The scope of these binders is the process P. We use the standard notions of *free* and *bound names* of processes, denoted as $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$ respectively, and of α-equivalence.

2.1 The Reduction Semantics

We present here the reduction semantics for our calculus. We give it in two steps, following the approach of Milner [13], separating the structural congruence that governs the static relations among processes from the reductions that rule their interactions. A structural congruence relation equates all the processes we do not want to distinguish. It is introduced as a small collection of axioms that allow to manipulate the structure of processes. This relation is intended to express some basic facts about the operators, such as commutativity of parallel composition. The second step is defining the way in which processes evolve dynamically by means of an operational semantics. We simplify the second step by closing the allowed transitions w.r.t. the structural congruence.

Definition 1 (Structural congruence). *The* structural congruence $\equiv$ *is the least congruence satisfying the abelian monoid laws for parallel composition (associativity, commutativity and* $\mathbf{0}$ *as identity) and commutativity of choice, and which is closed under* α*-renaming and under the following axioms:*

1. *Scope laws:*

$$(u)\mathbf{0} \equiv \mathbf{0}, \qquad (u)(v)P \equiv (v)(u)P,$$
$$P \mid (u)Q \equiv (u)(P \mid Q)\ , \quad \text{if } u \notin \mathrm{fn}(P)$$
$$\langle\!|(z)P\ ;\ Q|\!\rangle_x \equiv (z)\langle\!|P\ ;\ Q|\!\rangle_x\ , \quad \text{if } z \notin \{x\} \cup \mathrm{fn}(Q)$$

2. *Workunit laws:*

$$\langle\!|\mathbf{0}\ ;\ Q|\!\rangle_x \equiv \mathbf{0}$$
$$\langle\!|\langle\!|P\ ;\ Q|\!\rangle_y \mid R\ ;\ S|\!\rangle_x \equiv \langle\!|P\ ;\ Q|\!\rangle_y \mid \langle\!|R\ ;\ S|\!\rangle_x$$

3. Floating law:

$$\langle\!|\overline{z}\,\widetilde{u} \mid P \;;\; Q|\!\rangle_x \equiv \overline{z}\,\widetilde{u} \mid \langle\!|P \;;\; Q|\!\rangle_x$$

The scope laws are standard while novelties regard workunit and floating laws. The law $\langle\!|\mathbf{0} \;;\; Q|\!\rangle_x \equiv \mathbf{0}$ defines a committed workunit, namely a workunit with $\mathbf{0}$ as body. Such a workunit cannot fail anymore and thus it is equivalent to $\mathbf{0}$. The law $\langle\!|\langle\!|P \;;\; Q|\!\rangle_y \mid R \;;\; S|\!\rangle_x \equiv \langle\!|P \;;\; Q|\!\rangle_y \mid \langle\!|R \;;\; S|\!\rangle_x$ moves workunits outside parents, thus flattening the nesting. Notwithstanding this flattening, parent workunits may still affect children, but this has to be programmed explicitly, exploiting the available communication primitives. The law $\langle\!|\overline{z}\,\widetilde{u} \mid P \;;\; Q|\!\rangle_x \equiv \overline{z}\,\widetilde{u} \mid \langle\!|P \;;\; Q|\!\rangle_x$ floats messages outside workunit boundaries. By this law, messages are particles that independently move towards their inputs. The intended semantics is the following: if a process emits a message, this message traverses the surrounding workunit boundaries until it reaches the corresponding input. In case an outer workunit fails, recovery for this message may be detailed inside the handler process. When a workunit fails we will take care of messages and other workunits inside it (which may also have been included by applying the structural axioms above in the opposite direction), and preserve them.

The dynamic behavior of processes is defined by the reduction relation below, where we use the shortcut:

$$\langle\!|P \;;\; Q|\!\rangle \stackrel{\text{def}}{=} (z)\langle\!|P \;;\; Q|\!\rangle_z \text{ where } z \notin \text{fn}(P) \cup \text{fn}(Q)$$

Definition 2 (Reduction semantics). *The* reduction relation $\rightarrow$ *is the least relation satisfying the axioms below, and closed under* $\equiv$ *and under the contexts* $(x)\_\,,\; \_ \mid R$, *and* $\langle\!|\,\_ \;;\; R|\!\rangle_z$*:*

$$\overline{x_i}\,\widetilde{v} \mid \textstyle\sum_{i\in I} x_i(\widetilde{u_i}).P_i \;\rightarrow\; P_i\{\widetilde{v}/\widetilde{u_i}\} \quad \text{(R-COM)}$$

$$\overline{x}\,\widetilde{v} \mid !x(\widetilde{u}).P \;\rightarrow\; P\{\widetilde{v}/\widetilde{u}\} \mid !x(\widetilde{u}).P \quad \text{(R-REP)}$$

$$\overline{x} \mid \langle\!|\textstyle\prod_{i\in I}\sum_{s\in S_i} x_{i,s}(\widetilde{u_{i,s}}).P_{i,s} \mid \prod_{j\in J} !x_j(\widetilde{u_j}).P_j \;;\; Q|\!\rangle_x \;\rightarrow\; \langle\!|Q \;;\; \mathbf{0}|\!\rangle \quad \text{(R-FAIL)}$$

where $J \neq \emptyset \vee I \neq \emptyset, S_i \neq \emptyset$

Rules (R-COM) and (R-REP) are standard in process calculi and they model input-output interaction and lazy replication. Rule (R-FAIL) models workunit failures: when a unit aborts (receiving an empty message on its abort port), the corresponding body is terminated and the handler activated. On the contrary, aborts are not possible if the transaction is already terminated (namely every thread in the body has completed its own work). For this reason, when the handler is activated, we close the workunit by restricting its name. The reason to maintain the structure will be clear in the section relative to the labeled semantics (Section 3).

2.2 The Extensional Semantics

The extensional semantics of $\texttt{web}\pi_\infty$ relies on the notions of barb and context. We say that P has a *barb* at x, and write $P \downarrow x$, if P manifests an output on the free name x.

Definition 3. *We define $P \downarrow x$ as the least relation satisfying the rules:*

$$\begin{array}{cc} \overline{x}\,\widetilde{u} \downarrow x & \\ (z)P \downarrow x & \text{if } P \downarrow x \text{ and } x \neq z \\ P \,|\, Q \downarrow x & \text{if } P \downarrow x \text{ or } Q \downarrow x \\ \langle\!|P \,;\, Q|\!\rangle_z \downarrow x & \text{if } P \downarrow x \end{array}$$

It is worth to notice that inputs (both simple and replicated) have no barb. This is standard in asynchronous calculi and represents the fact that an observer has no direct way of knowing whether the message (s)he has sent has been received.

Definition 4. Process contexts*, noted $\mathsf{C}_\pi[\cdot]$, are defined by the following grammar:*

$$\mathsf{C}_\pi[\cdot] ::= [\cdot] \mid (x)\mathsf{C}_\pi[\cdot] \mid \mathsf{C}_\pi[\cdot]|P \mid \textstyle\sum_{i\in I} x_i(\widetilde{u_i}).P_i + x(\widetilde{u}).\mathsf{C}_\pi[\cdot] \mid\, !x(\widetilde{u}).\mathsf{C}_\pi[\cdot] \mid \langle\!|\mathsf{C}_\pi[\cdot] \,;\, P|\!\rangle_x \mid \langle\!|P \,;\, \mathsf{C}_\pi[\cdot]|\!\rangle_x$$

Barbed bisimilarity is usually defined as the largest bisimulation on the reduction relation such that the equated terms have the same barbs. Usually, such a relation is not a congruence and the barbed congruence is defined as the maximal barbed bisimulation that is also a congruence. In the following $\rightarrow_n$ stands for $\underbrace{\rightarrow \ldots \rightarrow}_{n}$. We write $\Rightarrow$ to denote $\rightarrow_n$ for some $n \geq 0$. We also write $P \Downarrow x$ for $\exists P'.P \Rightarrow P' \wedge P' \downarrow x$.

Definition 5 (Barbed congruence). *A* barbed bisimulation *is a symmetric binary relation $\mathcal{S}$ between processes such that $P\,\mathcal{S}\,Q$ implies*

1. *if $P \downarrow x$ then $Q \Downarrow x$;*
2. *if $P \rightarrow P'$ then $Q \Rightarrow Q'$ and $P'\,\mathcal{S}\,Q'$.*

Barbed congruence*, denoted as $\approx$, is the largest barbed bisimulation that is also a congruence.*

3 The Labeled Semantics

Barbed congruence requires quantification over all contexts, thus making direct proofs particularly difficult. A standard device to avoid such a quantification consists in introducing a labeled operational model and equipping it with a weak (asynchronous) bisimulation. If one can prove that bisimulation implies barbed congruence, then it can be used as a useful proof technique for behavioral equivalence.

We use some auxiliary machineries: the *extraction function* $\mathsf{xtr}(P)$, that extracts messages and units out of the process P, and is needed to define the abort of a unit:

$$\begin{aligned}
\mathsf{xtr}(\mathbf{0}) &= \mathbf{0}\\
\mathsf{xtr}(\overline{x}\,\widetilde{v}) &= \overline{x}\,\widetilde{v}\\
\mathsf{xtr}(\textstyle\sum_{i\in I} x_i(\widetilde{u_i}).P_i) &= \mathbf{0}\\
\mathsf{xtr}((x)P) &= (x)\mathsf{xtr}(P)\\
\mathsf{xtr}(P \mid Q) &= \mathsf{xtr}(P) \mid \mathsf{xtr}(Q)\\
\mathsf{xtr}(!x(\widetilde{u}).P) &= \mathbf{0}\\
\mathsf{xtr}(\langle\!|P\ ;\ Q|\!\rangle_x) &= \langle\!|P\ ;\ Q|\!\rangle_x
\end{aligned}$$

and the input predicate $\mathsf{inp}(P)$, which verifies whether a process contains an input that is not inside a workunit, which is used to find out whether a unit is still active. It is the least relation such that:

$$\begin{array}{ll}
\mathsf{inp}(\textstyle\sum_{i\in I} x_i(\widetilde{u_i}).P_i) & \\
\mathsf{inp}((x)P) & \text{if } \mathsf{inp}(P)\\
\mathsf{inp}(P \mid Q) & \text{if } \mathsf{inp}(P) \text{ or } \mathsf{inp}(Q)\\
\mathsf{inp}(!x(\widetilde{u}).P) &
\end{array}$$

In this section it will be useful to have clear the following property:

Proposition 1. *The extraction function is idempotent, i.e., if P is a* $\mathtt{web}\pi_\infty$ *process then* $\mathsf{xtr}(P) = \mathsf{xtr}(\mathsf{xtr}(P))$.

Proof. The proof is by structural induction on P. All the cases are straightforward. □

We can now define the labeled semantics. Let μ range over input labels $x(\widetilde{u})$, bound output labels $(\widetilde{z})\overline{x}\,\widetilde{u}$ where $\widetilde{z} \subseteq \widetilde{u}$, and the label τ. Let also $\mathrm{fn}(\tau) = \emptyset$, $\mathrm{fn}(x(\widetilde{u})) = \{x\}$, $\mathrm{fn}(\overline{x}\,\widetilde{u}) = \{x\} \cup \widetilde{u}$, and $\mathrm{fn}((\widetilde{z})\overline{x}\,\widetilde{u}) = \{x\} \cup \widetilde{u} \setminus \widetilde{z}$. Finally, let $\mathrm{bn}(\mu)$ be $\widetilde{z}$ if $\mu = (\widetilde{z})\overline{x}\,\widetilde{u}$, $\widetilde{u}$ if $\mu = x(\widetilde{u})$, and $\emptyset$ otherwise. We implicitly identify terms up to α-renaming $\equiv_\alpha$, that is, if $P \equiv_\alpha Q$, $P' \equiv_\alpha Q'$ and $P \xrightarrow{\mu} P'$ then $Q \xrightarrow{\mu} Q'$. In the following we will use again the shortcut:

$$\langle\!|P\ ;\ Q|\!\rangle \stackrel{\text{def}}{=} (z)\langle\!|P\ ;\ Q|\!\rangle_z \text{ where } z \notin \mathrm{fn}(P) \cup \mathrm{fn}(Q)$$

Definition 6 (Labeled semantics). *The* transition relation *of* $\mathtt{web}\pi_\infty$ *processes, noted* $\xrightarrow{\mu}$*, is the least relation satisfying the rules:*

$$\text{(IN)}\quad \textstyle\sum_{i\in I} x_i(\widetilde{u_i}).P_i \xrightarrow{x_i(\widetilde{u_i})} P_i \qquad \text{(OUT)}\quad \overline{x}\,\widetilde{u} \xrightarrow{\overline{x}\,\widetilde{u}} \mathbf{0} \qquad \text{(REPIN)}\quad !x(\widetilde{u}).P \xrightarrow{x(\widetilde{u})} P \mid !x(\widetilde{u}).P$$

$$\text{(RES)}\ \frac{P \xrightarrow{\mu} P' \quad x \notin \mathrm{fn}(\mu) \cup \mathrm{bn}(\mu)}{(x)P \xrightarrow{\mu} (x)P'} \qquad \text{(OPEN)}\ \frac{P \xrightarrow{(\widetilde{v})\overline{x}\,\widetilde{u}} P' \quad w \neq x \quad w \in \widetilde{u}\setminus\widetilde{v}}{(w)P \xrightarrow{(w\widetilde{v})\overline{x}\,\widetilde{u}} P'}$$

$$\text{(PAR)}\ \frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\mu} P' \mid Q}$$

$$\text{(COM)}\ \frac{P \xrightarrow{(\widetilde{w})\overline{x}\,\widetilde{v}} P' \quad Q \xrightarrow{x(\widetilde{u})} Q' \quad \widetilde{w} \cap \mathrm{fn}(Q) = \emptyset \quad |\widetilde{v}| = |\widetilde{u}|}{P \,|\, Q \xrightarrow{\tau} (\widetilde{w})(P' \,|\, Q'\{\widetilde{v}/\widetilde{u}\})}$$

$$\text{(ABORT)}\ \frac{\mathsf{inp}(P)}{\langle\!|P \,;\, Q|\!\rangle_x \xrightarrow{x()} \langle\!|\mathsf{xtr}(P) \,|\, Q \,;\, \mathbf{0}|\!\rangle} \qquad \text{(SELF)}\ \frac{P \xrightarrow{\overline{x}} P' \quad \mathsf{inp}(P)}{\langle\!|P \,;\, Q|\!\rangle_x \xrightarrow{\tau} \langle\!|\mathsf{xtr}(P') \,|\, Q \,;\, \mathbf{0}|\!\rangle}$$

$$\text{(WUNIT)}\ \frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\mu) \cap (\mathrm{fn}(Q) \cup \{x\}) = \emptyset}{\langle\!|P \,;\, Q|\!\rangle_x \xrightarrow{\mu} \langle\!|P' \,;\, Q|\!\rangle_x}$$

Rules involving parallel composition have mirror cases that we have omitted.

The first seven rules are standard in π-calculus. We just remind the role of the bound output $(u)\overline{x}\,u$ in $P \xrightarrow{(u)\overline{x}\,u} Q$. This kind of action means that P emits a private name u (a name bound in P) on the port x. Bound output actions arise from free output actions which carry names out of their scope as in the process $(u)\overline{x}\,u$. Let us discuss the rules related to workunits. Rule (WUNIT) is the simplest one: it lifts transitions to workunit contexts modeling the evolution of the body. In this sense it is very similar, for instance, to rules (PAR) and (RES). Rule (ABORT) models transaction termination due to an abort message. The premise checks that the unit body is still alive – it contains an active input – and, in this case, the compensation Q is triggered. We carefully do not erase the messages and the units in the body, which are extracted using the function $\mathsf{xtr}(\cdot)$. We remark that abort is not possible if the unit body P has completed, namely $\mathsf{inp}(P)$ is false. Rule (SELF) is similar to (ABORT), taking into account the case when the abort message is raised by the body of the unit. In this case, the handler Q can be spawned only if the body P cannot commit, i.e. if some input-guarded process is still waiting inside the process after the signaling of x.

Finally, two remarks deserve to be made: the first one concerns the shortcut $\langle\!|P \,;\, Q|\!\rangle$. This shortcut is used in rules (ABORT) and (SELF) to preserve the workunit structure after its abort. This could appear to be a strange design choice because this structure could be considered a redundant information once the workunit has aborted. Instead, it is important to retain it to have the input predicate falsity stable w.r.t. the transition relation. Indeed, it is not reasonable that if $\neg\mathsf{inp}(P)$ and $P \xrightarrow{\mu} P'$ then $\mathsf{inp}(P')$, since this corresponds to undo a commit. Note that the opposite instead makes sense, i.e., if $\mathsf{inp}(P)$ and $P \xrightarrow{\mu} P'$ then $\neg\mathsf{inp}(P')$ (for example in $\overline{x} \,|\, x().\mathbf{0}$), since this models a commit. However, the proposition below shows that the input predicate is stable under output transitions, i.e., a process can never commit via an output.

The second remark regards the side condition $\mathsf{inp}(P)$ in the rule (SELF). It should be written $\mathsf{inp}(P')$, referring to the pending state of some input in the process P' *after* the x signal. Usually, it is not very elegant and it is not a common practice in transition systems to write down a side condition related to

the right side of a premise. Anyway, it is safe to write $\mathsf{inp}(P)$ instead of $\mathsf{inp}(P')$. We will prove this fact with the following:

Proposition 2. *Let P be a $\mathtt{web}\pi_\infty$ process:*

1. *if $P \xrightarrow{\overline{x}\,\widetilde{u}} Q$ and $\mathsf{inp}(P)$ then $\mathsf{inp}(Q)$*
2. *if $\neg\mathsf{inp}(P)$ and $P \xrightarrow{\mu} Q$ then $\neg\mathsf{inp}(Q)$.*

Proof. We give just a brief sketch of the proof because of space constraints. Both the parts of the proof are by structural induction on P. In the first case one just has to consider the cases where $\mathsf{inp}(P)$, while in the second one the other cases have to be considered. □

3.1 Weak Asynchronous Bisimilarity

Recalling the weak asynchronous bisimilarity presented in [1] we define a weak asynchronous bisimilarity for $\mathtt{web}\pi_\infty$. We then find a suitable variant, that we call *closed bisimilarity*, which can be used as a tool to prove weak barbed congruence.

Definition 7 (Weak asynchronous bisimilarity). *We define $\stackrel{\tau}{\Rightarrow}$ as the reflexive and transitive closure of $\xrightarrow{\tau}$ and $\stackrel{\mu}{\Rightarrow}$ as $\stackrel{\tau}{\Rightarrow}\xrightarrow{\mu}\stackrel{\tau}{\Rightarrow}$.*

A weak asynchronous bisimulation *is a symmetric binary relation $\mathcal{R}$ such that $P\,\mathcal{R}\,Q$ implies:*

1. *if $P \xrightarrow{\tau} P'$, then $Q \stackrel{\tau}{\Rightarrow} Q'$ and $P'\,\mathcal{R}\,Q'$;*
2. *if $P \xrightarrow{(\widetilde{z})\overline{x}\,\widetilde{u}} P'$ and $\widetilde{z} \cap \mathrm{fn}(Q) = \emptyset$, then $Q \stackrel{(\widetilde{z})\overline{x}\,\widetilde{u}}{\Longrightarrow} Q'$ and $P'\,\mathcal{R}\,Q'$;*
3. *if $P \xrightarrow{x(\widetilde{u})} P'$ then*
   (a) *either $Q \stackrel{x(\widetilde{u})}{\Longrightarrow} Q'$, and $P'\,\mathcal{R}\,Q'$;*
   (b) *or $Q \stackrel{\tau}{\Rightarrow} Q'$, and $P'\,\mathcal{R}\,(Q'|\overline{x}\,\widetilde{u})$.*

Weak asynchronous bisimilarity $\dot{\approx}_a$ is the largest weak asynchronous bisimulation.

Unfortunately $\dot{\approx}_a$ is not a congruence as it is instead in asynchronous π-calculus [16]. To show this fact consider the following counterexample. Let

$$P \stackrel{\text{def}}{=} \mathbf{0}$$
$$Q \stackrel{\text{def}}{=} (z)z()$$

then $P \;\dot{\approx}_a\; Q$ because they both cannot move. As you can easily see $\mathsf{inp}(Q)$ holds but $\mathsf{inp}(P)$ does not, so if you consider the context $\langle\!|\, \mathsf{C}_\pi[\cdot]\ ;\ \overline{y}\,|\!\rangle_x$ and the rule (ABORT) you can see that the processes

$$\langle\!|\, \mathbf{0}\ ;\ \overline{y}\,|\!\rangle_x$$
$$\langle\!|\, (z)z(\,);\ \overline{y}\,|\!\rangle_x$$

behave differently with respect to the asynchronous bisimilarity definition given above. To solve this problem and have an equivalence which is also a congruence it is necessary to close it under the input predicate according to the following definition:

Definition 8. *A binary relation $\mathcal{R}$ over processes is input predicate-closed if $P\,\mathcal{R}\,Q$ implies $\mathsf{inp}(P) = \mathsf{inp}(Q)$.*

Unfortunately this is not enough to get a congruence. Consider now the counterexample:

$$P \stackrel{\text{def}}{=} !x().y() \mid \langle\!| z().u() \;;\; \mathbf{0} |\!\rangle$$
$$Q \stackrel{\text{def}}{=} z().u() \mid \langle\!| !x().y() \;;\; \mathbf{0} |\!\rangle$$

P and Q behave in the same way in the sense that $P \,\dot{\approx}_a\, Q$. Also it is easy to see that $\mathsf{inp}(P) = \mathsf{inp}(Q)$ but, unfortunately, $\mathsf{xtr}(P)$ and $\mathsf{xtr}(Q)$ are not bisimilar, since $\mathsf{xtr}(P) = \langle\!| z().u() \;;\; \mathbf{0} |\!\rangle$ and $\mathsf{xtr}(Q) = \langle\!| x().y() \;;\; \mathbf{0} |\!\rangle$, thus the two processes behave differently when inserted, e.g., in the context $\langle\!| \cdot \;;\; \mathbf{0} |\!\rangle_x$. To solve this problem we also need an additional definition:

Definition 9. *A binary relation $\mathcal{R}$ over processes is extract-closed if $P\,\mathcal{R}\,Q$ implies $\mathsf{xtr}(P)\,\mathcal{R}\,\mathsf{xtr}(Q)$.*

Now, we can define a labeled bisimilarity, that we call *closed bisimilarity*, and prove that it is a congruence.

Definition 10 (Closed bisimilarity). Closed bisimilarity $\approx_a$ *is the largest weak asynchronous bisimulation that is input predicate-closed and extract-closed.*

We study now some properties of closed bisimilarity.

Theorem 1. *Closed bisimilarity $\approx_a$ is a congruence, i.e. given two processes P and Q such that $P \approx_a Q$ then $\mathsf{C}_\pi[P] \approx_a \mathsf{C}_\pi[Q]$ for each context $\mathsf{C}_\pi[\cdot]$.*

Proof. The proof is by structural induction over contexts, and each case requires a coinduction. Because of space constraints we give only the proof for workunit body and handler, the other cases being anyway similar to the corresponding cases of the analogous theorem for the asynchronous π-calculus (see [16]).

For the body we have to prove that $P \approx_a Q$ implies $\langle\!| P \;;\; R |\!\rangle_x \approx_a \langle\!| Q \;;\; R |\!\rangle_x$. Let us consider the three relevant cases of the definition. In the first case (rule (ABORT)), if $\langle\!| P \;;\; R |\!\rangle_x \xrightarrow{x()} \langle\!| \mathsf{xtr}(P) \mid R \;;\; \mathbf{0} |\!\rangle$ we must have $\mathsf{inp}(P)$ and, for the input closure, also $\mathsf{inp}(Q)$. Thus $\langle\!| Q \;;\; R |\!\rangle_x \xrightarrow{x()} \langle\!| \mathsf{xtr}(Q) \mid R \;;\; \mathbf{0} |\!\rangle$ and the statement follows from the coinductive hypothesis and the extract closure. The second case (rule (SELF)) is $\langle\!| P \;;\; R |\!\rangle_x \xrightarrow{\tau} \langle\!| \mathsf{xtr}(P') \mid R \;;\; \mathbf{0} |\!\rangle$ if $P \xrightarrow{\overline{x}} P'$. This also requires $\mathsf{inp}(P)$. This implies $\mathsf{inp}(Q)$ because of the input closure. Thus, since $P \approx_a Q$ we have $Q \stackrel{\overline{x}}{\Rightarrow} Q'$ and $P' \approx_a Q'$. Using rule (WUNIT) to lift the τ steps and rule (SELF) for the $\overline{x}$ step we get $\langle\!| Q \;;\; R |\!\rangle_x \stackrel{\tau}{\Rightarrow} \langle\!| \mathsf{xtr}(Q') \mid R \;;\; \mathbf{0} |\!\rangle$. Note, in fact, that, since outputs are asynchronous, we can always suppose that all the τ actions are performed before the output, that is when the workunit is still able to participate to the interaction. The statement follows from the coinductive hypothesis and the extract closure. For the last case (rule (WUNIT)) the proof is trivial because we simply lift the behavior of the body to the workunit context.

For the handler we have to prove that $P \approx_a Q$ implies $\langle\!| R \ ; \ P |\!\rangle_x \approx_a \langle\!| R \ ; \ Q |\!\rangle_x$. In this case P and Q can move only when shifted to the body part, as it happens in rule (ABORT) and rule (SELF). Since they are moved without being changed, then the thesis follows by coinduction. □

We prove now some auxiliary lemmas that will bring us nearer to our main goal.

Lemma 1. *Let P be a $\mathtt{web}\pi_\infty$ process. Then the following holds:*

1. *P can always be written in the form:*

$$P \equiv (\widetilde{z})(\prod_{i\in I}\sum_{s\in S_i} x_{i,s}(\widetilde{u_{i,s}}).P_{i,s} \mid \prod_{l\in L}!x_l(\widetilde{u_l}).P_l \mid \prod_{j\in J}\langle\!| P_j \ ; \ Q_j |\!\rangle_{x_j} \mid \prod_{k\in K}\overline{x_k}\,\widetilde{u_k})$$

2. *$\mathsf{xtr}(P)$ can always be written in the form:*

$$\mathsf{xtr}(P) \equiv (\widetilde{z})(\prod_{j\in J}\langle\!| P_j \ ; \ Q_j |\!\rangle_{x_j} \mid \prod_{k\in K}\overline{x_k}\,\widetilde{u_k})$$

Proof. For the first part it is necessary to apply structural congruence rules: in particular workunit laws to flatten the workunit structure, floating laws to extract output particles outside of workunits, parallel and summation laws to rearrange the order of processes and scope laws to factorize names in $\widetilde{z}$. For the second part, notice that all the structural axioms commute with function $\mathsf{xtr}(\cdot)$, thus it is enough to put P in the normal form above and then apply the extract function. □

Lemma 2. *Let P be a $\mathtt{web}\pi_\infty$ process. Then the following holds:*

1. *$P \xrightarrow{\overline{x}\,\widetilde{u}} P'$ only if $\mathsf{xtr}(P) \neq \mathbf{0}$*
2. *$P \xrightarrow{\overline{x}\,\widetilde{u}} P'$ if and only if $\mathsf{xtr}(P) \xrightarrow{\overline{x}\,\widetilde{u}} \mathsf{xtr}(P')$*

Proof. Both the parts are by induction on the structure of P. The first part is trivial, let us consider the second one. Thanks to Lemma 1, we can always divide a process in two parallel components P_1 and P_2, such that P_1 can not perform outputs and $\mathsf{xtr}(P_1) = \mathbf{0}$, and P_2 can perform outputs (unless it is $\mathbf{0}$) and $\mathsf{xtr}(P_2) = P_2$. The thesis follows trivially. □

Now we need to define a new concept of *input context* which is auxiliary to the next lemma.

Definition 11. Input contexts*, noted $\mathsf{N}[\cdot]$, are defined by the following grammar:*

$$\begin{aligned}\mathsf{N}[\cdot] ::= & \textstyle\sum_{i\in I} x_i(\widetilde{u_i}).P_i + [\cdot](\widetilde{v}).P \\ & ![\cdot](\widetilde{v}).P \\ & \mathsf{N}[\cdot] \mid P \\ & (z)\mathsf{N}[\cdot] \\ & \langle\!| \mathsf{N}[\cdot] \ ; \ P |\!\rangle_z \\ & \langle\!| P \ ; \ Q |\!\rangle_{[\cdot]}\end{aligned}$$

Lemma 3. *Let P be a $\mathtt{web}\pi_\infty$ process. Then*

1. $P \xrightarrow{(\widetilde{z})\overline{x}\,\widetilde{u}} P'$ *implies* $P \equiv (\widetilde{z})(P' \mid \overline{x}\,\widetilde{u})$
2. $P \xrightarrow{x(\widetilde{u})} P'$ *implies* $P \equiv \mathsf{N}[x]$

Proof. For the first part the proof is by induction on the proof tree of $P \xrightarrow{(\widetilde{z})\overline{x}\,\widetilde{u}} P'$. The base case is when $\overline{x}\,\widetilde{u} \xrightarrow{\overline{x}\,\widetilde{u}} \mathbf{0}$ by the (OUT) rule and is trivial. The inductive cases are related to the rules (WUNIT), (PAR), (RES) and (OPEN). The proof is similar in all the cases. We just show the case of rule (WUNIT). By inductive hypothesis we know that $P \equiv (\widetilde{z})(P' \mid \overline{x}\,\widetilde{u})$. Then $\langle\!| P \ ; \ Q |\!\rangle_y \equiv (\widetilde{z})(\langle\!| P' \ ; \ Q |\!\rangle_y \mid \overline{x}\,\widetilde{u})$ using the floating law, as required.

For the second part the proof is by induction on the proof tree of $P \xrightarrow{x(\widetilde{u})} P'$. We have three base cases related to the rules (IN), (REPIN) and (ABORT). The cases follows directly by definition. The inductive cases are related to rules (RES), (PAR) and (WUNIT) and are trivial too. □

The next lemma analyzes the relations between reduction semantics and barbs on one side, and labeled transitions on the other side.

Lemma 4. *Let P be a $\mathtt{web}\pi_\infty$ process. Then*

1. $P \downarrow x$ *if and only if* $P \xrightarrow{(\widetilde{z})\overline{x}\,\widetilde{u}} Q$ *for some* Q, $\widetilde{z}$ *and* $\widetilde{u}$
2. $P \xrightarrow{\tau} Q$ *implies* $P \to Q$
3. $P \to Q$ *implies that there is* R *such that* $R \equiv Q$ *and* $P \xrightarrow{\tau} R$

Proof. We prove the three statements in the lemma separately.

1. Since barbs are preserved by structural congruence, the first part follows from Lemma 3.
2. We have to prove that $P \xrightarrow{\tau} Q$ implies $P \to Q$. The proof is by induction on the proof tree of $P \xrightarrow{\tau} Q$. The base cases are two and they are related to rules (SELF) and (COM), i.e., the rules that introduce the label τ in the tree. The inductive cases are instead related to all those rules that move the τ label from the premise to the conclusion of the inference, i.e. (WUNIT), (PAR) and (RES). For space reasons we describe only the workunit part. The base case follows from the first part of Lemma 3. For the inductive case we have to prove that $\langle\!| P \ ; \ Q |\!\rangle_z \xrightarrow{\tau} R$ implies $\langle\!| P \ ; \ Q |\!\rangle_z \to R$. The inductive case is when $P \xrightarrow{\tau} P'$ and $\langle\!| P \ ; \ Q |\!\rangle_z \xrightarrow{\tau} \langle\!| P' \ ; \ Q |\!\rangle_z$ for the (WUNIT) rule. In this case we can apply the inductive hypothesis obtaining $P \to P'$ and, since the reduction relation is closed under the workunit context, $\langle\!| P \ ; \ Q |\!\rangle_z \to \langle\!| P' \ ; \ Q |\!\rangle_z$.
3. We have to prove that $P \to Q$ implies that there is R such that $R \equiv Q$ and $P \xrightarrow{\tau} R$. The proof is by induction on the proof tree of $P \to Q$. The base cases are three and they are related to rules (R-COM), (R-REP) and (R-FAIL). The inductive cases are instead related to the closures under contexts and

structural congruence. We show only the workunit case: if $P \to P'$ we have $\langle\!| P \ ; \ Q |\!\rangle_z \to \langle\!| P' \ ; \ Q |\!\rangle_z$ for the the context closure of the reduction relation. By inductive hypothesis we also have $P \xrightarrow{\tau} R$ with $R \equiv P'$. From this fact, using rule (WUNIT) of the labeled semantics, we get $\langle\!| P \ ; \ Q |\!\rangle_z \xrightarrow{\tau} \langle\!| R \ ; \ Q |\!\rangle_z$ where $\langle\!| R \ ; \ Q |\!\rangle_z \equiv \langle\!| P' \ ; \ Q |\!\rangle_z$. □

Now we are ready to prove our main result, which shows that closed bisimilarity can be used as a tool to prove weak barbed congruence.

Theorem 2. *For each pair of* $\texttt{web}\pi_\infty$ *processes* P *and* Q, $P \approx_a Q$ *implies* $P \approx Q$.

Proof. Lemma 4 proved that $\approx_a$ is a weak barbed bisimulation. We have also proved (Theorem 1) that $\approx_a$ is a congruence. Since $\approx$ is the largest barbed bisimulation that is a congruence then the thesis follows. □

4 Relevant Examples

The theory developed so far allows us to prove interesting properties about $\texttt{web}\pi_\infty$ processes. In this section we show some examples of pattern reducibility proving them correct as far as weak barbed congruence is concerned, and using closed bisimilarity as technical tool. This also shows that closed bisimilarity, whose completeness has not been proved yet, can be applied in many interesting cases.

Handlers Reducibility. Let us consider the following processes where $x' \notin \mathrm{fn}(P) \cup \mathrm{fn}(Q), x' \neq x$.

$$\begin{array}{l} \langle\!| P \ ; \ Q |\!\rangle_x \\ (x')(\langle\!| P \ ; \ \overline{x'} |\!\rangle_x \mid \langle\!| x'().Q \ ; \ \mathbf{0} |\!\rangle) \end{array}$$

The following theorem states that any workunit can be rewritten in another unit where the handler consists of a single asynchronous output and all the remaining parts of the process are moved in a separate unit and activated when necessary.

Theorem 3. $\langle\!| P \ ; \ Q |\!\rangle_x \approx (x')(\langle\!| P \ ; \ \overline{x'} |\!\rangle_x \mid \langle\!| x'().Q \ ; \ \mathbf{0} |\!\rangle)$

Proof. The relation ϕ on $\texttt{web}\pi_\infty$ processes defined as follows is a closed bisimulation. Below we intend P and Q to range over all processes, and x, x' range over all the names such that $x' \notin \mathrm{fn}(P) \cup \mathrm{fn}(Q), x' \neq x$.

$$\begin{array}{l} \phi = \{(P,P)\} \cup \{(\langle\!| P \ ; \ Q |\!\rangle_x, (x')(\langle\!| P \ ; \ \overline{x'} |\!\rangle_x \mid \langle\!| x'().Q \ ; \ \mathbf{0} |\!\rangle))\} \\ \quad \cup \{(\langle\!| \mathsf{xtr}(P) \mid Q \ ; \ \mathbf{0} |\!\rangle, (x')(\langle\!| \mathsf{xtr}(P) \ ; \ \mathbf{0} |\!\rangle \mid \langle\!| Q \ ; \ \mathbf{0} |\!\rangle))\} \end{array}$$

The full proof requires to show that the three conditions for closed bisimilarity are satisfied. This is quite easy, and the interested reader can refer to [11]. The thesis then follows thanks to Theorem 2. □

Decoupling of Service and Recovery Logics. Let us consider another couple of processes where $y \notin \mathrm{fn}(!z(u).P \mid Q) \cup \{v\}$:

$$\langle\!|\, !z(u).P \mid Q \;;\; \overline{v} \,|\!\rangle_x$$
$$(y)(\langle\!|\, !z(u).P \;;\; \overline{y} \,|\!\rangle_x \mid \langle\!|\, Q \mid (w)w(u) \;;\; \overline{v} \,|\!\rangle_y)$$

The following theorem shows the way in which a pattern expressing the service logic $!z(u).P \mid Q$ and the recovery logic for that service (intended as a single asynchronous output because of the previous theorem) can be decoupled and written separately by means of two different workunits. The property can be read also in the opposite sense, showing how two different workunits can be coupled in a single one.

Theorem 4. $\langle\!|\, !z(u).P \mid Q \;;\; \overline{v} \,|\!\rangle_x \approx (y)(\langle\!|\, !z(u).P \;;\; \overline{y} \,|\!\rangle_x \mid \langle\!|\, Q \mid (w)w(u) \;;\; \overline{v} \,|\!\rangle_y)$

Proof. The proof is similar to the one above, considering now as ϕ:

$$\phi = \{(P,P)\} \cup \{\langle\!|\, !z(u).P \mid Q \;;\; \overline{v} \,|\!\rangle_x, (y)(\langle\!|\, !z(u).P \;;\; \overline{y} \,|\!\rangle_x \mid \langle\!|\, Q \mid (w)w(u) \;;\; \overline{v} \,|\!\rangle_y)\}$$

where we intend P and Q to range over all processes, and z, u, v, x and y range over all the names such that $y \notin \mathrm{fn}(!z(u).P \mid Q) \cup \{v\}$. The only trick is that the addition of the deadlocked component $(w)w(u)$ is needed to ensure that the input predicate is true on the right hand side, as necessary to simulate the left hand side. □

The applications above show that in some cases of interest closed bisimilarity allows to use quite easily writable relations, while using weak barbed congruence directly is far more complex.

5 Conclusion

In this paper we analyzed some semantic issues in the framework of $\mathtt{web}\pi_\infty$, a simple extension of the π-calculus with untimed long running transactions. A timed extension of $\mathtt{web}\pi_\infty$, called $\mathtt{web}\pi$, has been presented in [8] to meet the challenge of time in composition. There $\mathtt{web}\pi$ has been equipped with an explicit mechanism for time elapsing and timeout handling. Adding time allows to express another interesting aspect of systems. Remember however that if one is not interested in the timing details, timeouts can be simply expressed as choices between the normal behavior and the timeout behavior. Discussing the notion of orchestration without considering time constraints makes it possible to focus on information flow, message passing, concurrency and resource mobility. Also, it allows to have a more abstract view using the weak semantics, which does not make sense in the timed framework, and which is the desired level of abstraction in many cases. Notice for instance that processes in our sample applications are not equivalent according to a strong equivalence.

Another related calculus is c-join [3], which extends join calculus [6] with long running transactions and compensations. The main difference between $\mathtt{web}\pi_\infty$

and c-join is that in the latter the nesting of transactions matters, since when the external transaction is aborted all the internal transactions are aborted too. This forces a particular way to deal with related transactions, while in our case this decision can be taken in a case by case way, by explicit sending abort signals to the other transactions. Note that in c-join instead a process can only abort the innermost transaction containing it (but the compensation can be programmed to propagate the abort to the upper level). Finally, in c-join, communication between processes in different transactions causes the transactions to be merged.

Long running transactions have been analyzed also using Compensating CSP [4], but this approach is more focused on the definition of compensations for large processes starting from definitions of compensations for their components, and it provides neither synchronization (apart from sequential composition) nor mobility.

This work contributes with a powerful and expressive language, with a solid semantics, that allows formal reasoning. The language shows a clear relation with the π-calculus and the actual encoding is a feasible task, while it would be quite harder to get such an encoding for XLANG and other web services composition languages. Future developments building on the results achieved in this paper include software tools for static analysis of programs based on composition of services. A useful result we achieved in [11] that stem from this work is a streamlined definitions of syntax and semantics of BPEL, to get a simpler way to model involved transaction behaviors. The overall goal of these works is to allow for improvement of quality and applicability of real composition languages.

Acknowledgments. The authors would like to strongly acknowledge Cosimo Laneve for his huge support and contribution and Luiz Olavo Bonino Da Silva Santos, Roberto Bruni and Hernán Melgratti for theirs comments.

References

1. R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theoret. Comput. Sci.*, 195(2):291–324, 1998.
2. A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C. K. Liu, V. Mehta, S. Thatte, P. Yendluri, A. Yiu, and A Alves. Web services business process execution language version 2.0. Technical report, Oasis, December 2005. Working draft.
3. R. Bruni, H. C. Melgratti, and U. Montanari. Nested commits for mobile calculi: Extending join. In *Proc. of IFIP TCS'04*, pages 563–576. Kluwer Academics, 2004.
4. M. J. Butler, C. A. R. Hoare, and C. Ferreira. A trace semantics for long-running transactions. In *25 Years Communicating Sequential Processes*, volume 3525 of *Lect. Notes in Comput. Sci.*, pages 133–150. Springer, 2004.
5. F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *Commun. ACM*, 46(10):29–34, 2003.
6. C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. of POPL'96*, pages 372–385. ACM Press, 1996.

7. M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.
8. C. Laneve and G. Zavattaro. Foundations of web transactions. In *Proc. of FoSSaCS'05*, volume 3441 of *Lect. Notes in Comput. Sci.*, pages 282–298. Springer, 2005.
9. F. Leymann. Web services flow language (WSFL 1.0). Technical report, IBM, May 2001.
10. R. Lucchi and M. Mazzara. A π-calculus based semantics for WS-BPEL. *J. Log. Algebr. Program.*, 2006. To appear.
11. M. Mazzara. *Towards Abstractions for Web Services Composition.* PhD thesis, Department of Computer Science, University of Bologna, 2006. Also available as Technical Report UBLCS-2006-08.
12. Microsoft BizTalk. `http://www.microsoft.com/biztalk/default.mspx`.
13. R. Milner. Functions as processes. *Math. Struct. in Comput. Sci.*, 2(2):119–141, 1992.
14. R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Inform. and Comput.*, 100(1):1–40,41–77, 1992.
15. C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003.
16. D. Sangiorgi and D. Walker. *The π-calculus: A theory of Mobile Processes.* Cambridge University Press, 2001.
17. C. Szyperski. *Component Software: Beyond Object-Oriented Programming, 2nd Ed.* Addison-Wesley/ACM Press, 2002.
18. S. Thatte. XLANG: Web services for businnes process design. Technical report, Microsoft Corporation, 2001. Downloadable from `www.gotdotnet.com/team/xml/wsspecs/xlang-c`.