# Timing Issues in Web Services Composition

Manuel Mazzara

Department of Computer Science, University of Bologna, Italy
mazzara@cs.unibo.it

**Abstract.** webπ is a recent process calculus introduced to formally specify Web Services composition. It extends the π-calculus with timed workunits, namely an asynchronous and temporized mechanism for events raising and catching. In this paper we encode Berger-Honda Timed-π in webπ timed workunits and we prove a simulation theorem. The overall perspective of this work is to make webπ comparable with both real composition languages and well established models for distributed components.

## 1 Introduction

Service Oriented Computing (SOC) is an emerging paradigm for distributed computing and e-business processing that finds its origin in object-oriented and component computing. Web services technology is a widespread accepted instantiation of SOC which should facilitate integration of newly built and legacy applications both within and across organizational boundaries avoiding difficulties due to different platform, heterogeneous programming languages, security firewall, etc... Exploiting this kind of ubiquitous network fabric would result in an increase of productivity and in a reduction of costs in B2B processes [17]. The idea behind this approach is to allow independently developed applications to be exposed as services and interconnected exploiting the already set up Web infrastructure with relative standards (HTTP [31], XML [12], SOAP [7] and WSDL [11]). These technologies, related to develop basic services and interconnect them on a point-to point basis, can be considered well established but B2B processing requires managing complex interactions involving a large number of participants and none of the above standards are able to meet this need. The way to build complex services out of simpler ones is called *composition* and it is still an open challenge [28].

Different organizations are presently working on composition proposals. The most important in the past have been IBM's WSFL [21] and Microsoft's XLANG [29]. These two have then converged in Web Services Business Process Execution Language [3] (WS-BPEL or BPEL for short) which is presently a working draft by OASIS. Another recent proposal in phase of standardization by the World Wide Web Consortium (W3C) is WS-CDL [18]. Both allow the definition of workflow-based composition of services with some similarities and some differences. Describing in details a synopsis between these two proposals is beyond the scope of this paper, however in section 2 some points will be sketched.

M. Bravetti et al. (Eds.): EPEW 2005 and WS-FM 2005, LNCS 3670, pp. 287–302, 2005.

### 1.1 Need for Foundations

XLANG, WS-BPEL and WS-CDL are claimed to be based on formal models (the $\pi$-calculus or its variant) to allow rigorous mathematical reasoning. For example, WS-CDL authors explicitly state to be in some relation with fusions and solos. In particular, WS-CDL is built atop the Global Model formalism (as presented in [17]) which refers to a precise $\pi$-calculus variant: the Explicit Solos Calculus [13], the theory underlying the Fusion Machine (a virtual machine implementing in a distributed manner the $\pi$-calculus). However, despite all this hype, no interesting relations with process algebras have been so far emphasized (no conceptual tools for analysis and reasoning, no software verification). In this way any mathematical rigor becomes pointless.

$\texttt{web}\pi_\infty$ [24] has been introduced to fill this gap. It is a simple and conservative extension of the $\pi$-calculus where the original algebra is augmented with an operator for asynchronous events raising and catching in order to enable the programming of widely accepted error handling techniques (such as long running transactions and compensations) with a reasonable simplicity. The ability to handle time is also considered a very appropriate feature when programming transactions where business services cannot wait forever for the reply of other parties. For this reason, $\texttt{web}\pi_\infty$ has a timed counterpart, $\texttt{web}\pi$ [19], which allows events to be temporized, i.e. to happen not only when processes explicitly raise them but also when timers expire. We address the problem of composing services starting directly from the $\pi$-calculus and considering our proposals as foundational models for composition simply to verify statements regarding any mathematical foundations of composition languages and not to say that the $\pi$-calculus is more suitable than other models (such as Petri nets) for these purposes. For an ongoing discussion about these foundational aspects refer to [30].

### 1.2 Error Handling and Web Transactions

Loosely coupled components like Web services, being autonomous in their decisions, may refuse requests and suspend their functionality without notice, thus making their behavior unreliable to other activities. Henceforth, most of the web languages also include the notion of loosely coupled transaction – called *web transaction* [22] in the following – as a unit of work involving loosely coupled activities that may last long periods of time. These transactions, being orthogonal to administrative domains, have the typical atomicity and isolation properties relaxed, and instead of assuming a perfect roll-back in case of failure, support the explicit programming of compensation activities. Web transactions usually contain the description of three processes; the body, the failure handler, and the compensation.

The failure handler is responsible for reacting to events that occur during the execution of the body; when these events occur, the body is blocked and the failure handler is activated. The compensation, on the contrary, is installed when the body commits; it remains available for outer transactions to require

some undo of previously performed actions. BPEL and WS-CDL both use this approach. However, in [25,23] we showed that different mechanisms for error handling are not necessary and we presented the BPEL semantics in terms of $\texttt{web}\pi_\infty$ which is based on the idea of event notification as the unique error handling mechanism. The same is feasible considering WS-CDL. This result allows us to extend any semantic considerations about $\texttt{web}\pi_\infty$ and $\texttt{web}\pi$ to BPEL and WS-CDL.

### 1.3 Contribution of the Paper

In [24] we used $\texttt{web}\pi_\infty$ as a theoretical and foundational model for web services composition and we proved its usefulness formalizing an e-commerce transactional scenario experimented in our preliminary work [14]. In those papers we did not address timing issues at all. We recognized the limits of those works and the usefulness of time handling when programming business transactions. For this reason, in this paper we consider *timed transactions*, i.e. transactions that can be interrupted by a timeout. Real workflow languages presently provide this feature: XLANG, for instance, includes a notion of timed transaction as a special case of long running activity. BPEL also allows similar behaviors by means of alarm clocks.

To meet the challenge of time in composition, $\texttt{web}\pi$ has been equipped with an explicit mechanism for time elapsing and timeout handling. Adding time we are able to express more meaningful and realistic scenarios in composition. The $\texttt{web}\pi$ model of time is inspired by Berger-Honda Timed-$\pi$ skipping the idle rule plus some minor variations. In this paper we present a synopsis of the two approaches underlying differences and similarities. We show the ability of $\texttt{web}\pi$ to cope with timing issues in a context of B2B web transactions proving that skipping the idle rule is not source of expressiveness loss. To do this we encode their time construct, called timer, in our timed workunit and we prove in detail a simulation theorem. This is intended as a major result of the paper and convinces us of the great flexibility of $\texttt{web}\pi$.

Another contribution stands in section 2 where we clarify some semantical aspects of composition languages and where we modify some terminology of $\texttt{web}\pi$ presenting detailed motivations. The overall perspective of this work is to make $\texttt{web}\pi$ comparable with both real composition languages and well established model for distributed components.

### 1.4 Related Work

Other papers discussing the formal semantics of compensable activities in this context are: the work by Hoare [15] which is mainly inspired by XLANG, the calculus of Butler and Ferreira [10] which is inspired by BPBeans, the $\pi$t-calculus [6] considering BizTalk and the work [8] dealing with short-lived transactions in BizTalk. The work in [9] also presents the formal semantics for a hierarchy of transactional calculi with increasing expressiveness.

### 1.5 Outline of the Paper

The paper is structured as follows: after the above introduction, section 2 tries to clarify some semantical aspects of composition languages and of our model. Section 3 presents webπ with its syntax and semantics while section 4 is devoted to an analogous description of the counterpart, $\pi_t$. The encoding of timers is showed and explained in section 5 where the correctness proof is also detailed and described. Finally, section 6 reports some conclusive considerations.

## 2 WS-BPEL, WS-CDL and webπ

It is worth noting that in this paper we are changing some terminology with respect to previous works presenting webπ [19,20] or webπ$_\infty$ [24,23]. In particular we are replacing the term *transaction* or *timed transaction* with the term *timed workunit* and the term *compensation* with the term *event handler*. This is because we believe that, using the old terminology and continuously associating webπ with real composition languages like WS-BPEL or WS-CDL, confusion and ambiguity can raise.

As explained in detail in [23], the WS-BPEL Recovery Framework has two different mechanisms for coping with abnormal situations: fault handler and compensation handler. Also WS-CDL provides mechanisms with a similar semantics called exceptions and finalizers. The basic wrapper containing operations and associated handlers is `scope` for WS-BPEL and `choreography` for WS-CDL. These mechanisms are thought to be used at different stages of computation: fault handling during the execution of an activity while compensation handling after its successful completion. While fault/exception handlers are typically provided by classical concurrent programming languages, compensation handlers or finalizers are peculiar to composition languages. Compensations are related with long running web transactions and the relative semantic deserves some attention.

It is important to remind that scopes and choreographies can be structured in a tree of nesting. Both WS-BPEL and WS-CDL allow compensations (or finalizer) to be available for a scope (or choreography) after its successful termination. BPEL has a constraint which forces a compensation to be triggered only by an enclosing scope which failed for some reason. WS-CDL, instead, allows a finalizer to be simply activated by a parent choreography which failed or not, without imposing particular constraints (motivation for this decision are related to speculative parallelism and can be found at [1,2]).

Anyway, compensation semantics is strictly about "partially reversing" of successful activities included in a "larger work" which failed. Differently, fault handling is a mechanism thought to interrupt "immediately" an activity when some abnormal situation happens. At that point, the normal execution is broken and no way to access the compensation handler is still available. After these considerations it is easy to see how webπ semantics is very far from the compensation semantics of composition languages. Indeed, webπ mechanism is more similar to fault handling. Anyway, we want to avoid to call it fault handler because we want to provide a foundational mechanism which is able, as already

showed, to encode both the presented mechanisms. In some sense, our work is close to the CORBA Activity Service Framework [16] which uses a similar event signalling mechanism. Both these approaches result more flexible with respect to WS-BPEL and WS-CDL semantics. For this reason we call it *event handler*. In fact, it is simply a generic framework for event handling and catching.

A last remark deserves to be made to clarify completely any possible objections. While WS-CDL does not support additional mechanisms except the two described above, WS-BPEL provides also a third mechanism called *event handler*, as in webπ. Its semantics, however, is different: a BPEL event handler listens to messages or alarm clock concurrently to the scope execution and handles all the events concurrently, even when multiple instances occur. If the scope terminates but some of these occurrences are still alive, they are allowed to normally terminate their execution. We showed that also this semantic can be encoded in webπ, so the presence of this additional machinery is not harmful. Anyway, it is important to underline that, although the names are equal, the behaviors are different.

We decided to adopt what we intend to be the more foundational mechanism to encode all the others and we gave it a name which was as general as possible. As a consequence of all these considerations, we changed also the term *transaction* in *workunit*, because, in general, a transaction is the composed effect of many workunits, not just a single one.

## 3 The Calculus webπ

webπ is a timed extension of the asynchronous π-calculus with an explicit wrapping constructor for activities and an associated event handler, developed in order to provide mathematical foundation for composition languages. Composition essentially describes workflow, with a particular emphasis on the communication aspects of loosely coupled activities, i.e. activities executed by remote, heterogeneous and independent services that could belong to different administrative domains, such as different companies.

### 3.1 webπ Syntax

The syntax of webπ relies on a countable set of *names*, ranged over by $x, y, z, u, w, s, s' \cdots$. Tuples of names are written $\widetilde{u}$. Natural numbers $\{0, 1, 2, 3, \cdots\}$ or $\infty$ are ranged over by $n, m, \cdots$. The set of *processes* is defined by the following syntax:

$$
\begin{array}{lll}
P ::= & & (\textbf{processes}) \\
& \mathbf{0} & (\text{nil}) \\
& \mid \overline{x}\,\langle \widetilde{u} \rangle & (\text{message}) \\
& \mid x(\widetilde{u}).P & (\text{input}) \\
& \mid (x)P & (\text{restriction}) \\
& \mid P \mid P & (\text{parallel composition}) \\
& \mid\, !x(\widetilde{u}).P & (\text{lazy replication}) \\
& \mid \langle\!| P \;;\; P |\!\rangle_s^n & (\text{timed workunit})
\end{array}
$$

A process can be the inert process $\mathbf{0}$, a message $\overline{x}\,\langle\widetilde{u}\rangle$ sent on a name $x$ that carries a tuple of names $\widetilde{u}$, an input $x(\widetilde{u}).P$ that consumes a message $\overline{x}\,\langle\widetilde{w}\rangle$ and behaves like $P\{^{\widetilde{w}}/_{\widetilde{u}}\}$, a restriction $(u)P$ that behaves as $P$ except that inputs and messages on $u$ are prohibited, a parallel composition of processes, a replicated input $!x(\widetilde{u}).P$ that consumes a message $\overline{x}\,\langle\widetilde{w}\rangle$ and behaves like $P\{^{\widetilde{w}}/_{\widetilde{u}}\} \mid !x(\widetilde{u}).P$ or a timed workunit $\langle\!|P\ ;\ R|\!\rangle_s^n$ that behaves as the *body* $P$ except that the *event handler* $R$ is triggered after $n$ steps or because the opportune abort signal $\overline{s}\,\langle\rangle$ is received. The label $n$ is called the *time stamp*. We remark that workunit names should be used with output capability only. For instance, it is not possible to write $s().P$. Our intuition is that workunit names are process identifiers, therefore two different workunits should never have the same name. Even if we conform with such intuition in this paper, we purposely do not enforce in $\texttt{web}\pi$ a discipline for the use of these names.

The calculus accounts for time by using positive natural numbers or $\infty$. The *timeless workunit* $\langle\!|P\ ;\ R|\!\rangle_s$ is an abbreviation for $\langle\!|P\ ;\ R|\!\rangle_s^\infty$, and we assume that $\infty + 1 = \infty$. Input $x(\widetilde{u}).P$, restriction $(x)P$ and lazy replication $!x(\widetilde{u}).P$ 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 alpha-equivalence, *free* and *bound names* of processes, noted $\texttt{fn}(P)$ and $\texttt{bn}(P)$, respectively. In particular, $\texttt{fn}(\langle\!|P\ ;\ R|\!\rangle_x^n) = \texttt{fn}(P) \cup \texttt{fn}(R) \cup \{x\}$ and alpha-equivalence equates $(x)(\langle\!|P\ ;\ Q|\!\rangle_x^n)$ with $(s)(\langle\!|P\{^s/_x\}\ ;\ Q\{^s/_x\}|\!\rangle_s^n)$.

### 3.2 The Reduction Semantics

We are now ready to introduce the formal specification of the semantics of $\texttt{web}\pi$. Following the tradition of $\pi$-calculus [26,27], we first define a structural congruence which equates all agents we will never want to distinguish for any semantic reason, and then use this when giving the operational semantics.

**Definition 1.** *The* structural congruence $\equiv$ *is the least congruence closed with respect to alpha-renaming, satisfying the abelian monoid laws for parallel (associativity, commutativity and* $\mathbf{0}$ *as identity), and the following axioms:*

1. *the scope laws:*

$$\begin{array}{rl} (u)\mathbf{0} \equiv \mathbf{0}, \qquad (u)(v)P \equiv (v)(u)P, & \\ P \mid (u)Q \equiv (u)(P \mid Q)\,, & \text{if } u \notin \texttt{fn}(P) \\ \langle\!|(z)P\ ;\ Q|\!\rangle_s^n \equiv (z)\langle\!|P\ ;\ Q|\!\rangle_s^n\,, & \text{if } z \notin \{s\} \cup \texttt{fn}(Q) \\ \langle\!|P\ ;\ (z)Q|\!\rangle_s^0 \equiv (z)\langle\!|P\ ;\ Q|\!\rangle_s^0\,, & \text{if } z \notin \{s\} \cup \texttt{fn}(P) \end{array}$$

2. *the repetition law:*

$$!x(\widetilde{u}).P \equiv x(\widetilde{u}).P \mid !x(\widetilde{u}).P$$

3. *the workunit laws:*

$$\begin{array}{c} \langle\!|\mathbf{0}\ ;\ Q|\!\rangle_x^s \equiv \mathbf{0} \\ \langle\!|\langle\!|P\ ;\ Q|\!\rangle_{s'}^n \mid R\ ;\ R'|\!\rangle_s^m \equiv \langle\!|P\ ;\ Q|\!\rangle_{s'}^n \mid \langle\!|R\ ;\ R'|\!\rangle_s^m \end{array}$$

*4. the floating laws:*

$$\langle\!|\overline{z}\,\langle\widetilde{u}\rangle \mid P \;;\; Q|\!\rangle_s^n \equiv \overline{z}\,\langle\widetilde{u}\rangle \mid \langle\!|P \;;\; Q|\!\rangle_s^n$$
$$\langle\!|y(\widetilde{v}).P \mid P' \;;\; \overline{z}\,\langle\widetilde{u}\rangle \mid Q|\!\rangle_s^0 \equiv \overline{z}\,\langle\widetilde{u}\rangle \mid \langle\!|y(\widetilde{v}).P \mid P' \;;\; Q|\!\rangle_s^0$$

The scope and repetition laws are almost standard: let us discuss workunit and floating laws. The law $\langle\!|\mathbf{0} \;;\; Q|\!\rangle_s^n \equiv \mathbf{0}$ defines committed workunits, namely those with $\mathbf{0}$ as body. These workunits, being committed, are equivalent to $\mathbf{0}$ and, therefore, cannot fail anymore. The law $\langle\!|\langle\!|P \;;\; Q|\!\rangle_{s'}^n \mid R \;;\; R'|\!\rangle_s^m \equiv \langle\!|P \;;\; Q|\!\rangle_{s'}^n \mid \langle\!|R \;;\; R'|\!\rangle_s^m$ moves workunits outside the parent, thus flattening the nesting. Notwithstanding this flattening, the parent can still affect the children by means of workunit names. The law $\langle\!|\overline{z}\,\langle\widetilde{u}\rangle \mid P \;;\; R|\!\rangle_s^n \equiv \overline{z}\,\langle\widetilde{u}\rangle \mid \langle\!|P \;;\; R|\!\rangle_s^n$ floats messages outside workunits, thus modelling the fact that messages are particles uploaded on the network as soon as they are emitted. The intended semantics is the following: if a process emits a message, this message traverses the surrounding boundaries, until it reaches the corresponding input. In case an outer workunit fails, recoveries for this message may be detailed inside the relative handler.

The main technical difficulty is time elapsing. In this model all the processes run on the same orchestrator, thus competing for the same processor time. We assume that every reduction costs one time slot. When a subprocess performs a reduction, the flow of time is communicated to all the running processes. This amounts to decrease the time stamps of the running timed workunits by 1, thus triggering handler processes of those that become dead. This operation is modelled by the *time stepper function* below, which is an accommodation to `web`$\pi$ of the corresponding function in [4]. The definition of this function and two other auxiliary functions are in order:

**the input predicate** $\mathtt{inp}(P)$**:** this predicate verifies whether a process contains an input that is not underneath a workunit. Formally:

$$\begin{array}{ll}\mathtt{inp}(x(\widetilde{u}).P) & \\ \mathtt{inp}((x)P) & \text{if } \mathtt{inp}(P) \\ \mathtt{inp}(P \mid Q) & \text{if } \mathtt{inp}(P) \text{ or } \mathtt{inp}(Q) \\ \mathtt{inp}(!x(\widetilde{u}).P) & \end{array}$$

**the time stepper function** $\phi(P)$**:** this function decreases the time stamp by 1 and is defined inductively in the following way:

$$\begin{array}{rl}\phi((x)P) = (x)\phi(P) & \\ \phi(P \mid Q) = \phi(P) \mid \phi(Q) & \\ \phi(\langle\!|P \;;\; R|\!\rangle_s^0) = \begin{cases}\langle\!|\phi(P) \;;\; \phi(R)|\!\rangle_s^0 & \text{if } \mathtt{inp}(P) \\ \langle\!|\phi(P) \;;\; R|\!\rangle_s^0 & \text{otherwise}\end{cases} & \\ \phi(\langle\!|P \;;\; R|\!\rangle_s^{n+1}) = \langle\!|\phi(P) \;;\; R|\!\rangle_s^n & \\ \phi(P) = P \qquad \text{otherwise} & \end{array}$$

All the preliminaries are in place for the definition of the reduction relation.

**Definition 2.** *The reduction relation $\rightarrow$ is the least relation satisfying the following reductions:*

$$\text{(COM)} \qquad \overline{x}\,\langle \widetilde{v}\rangle \,|\, x(\widetilde{u}).Q \rightarrow Q\{\widetilde{v}/\widetilde{u}\}$$

$$\text{(FAIL)}\ \overline{s}\,\langle\rangle \,|\, \langle\!|z(\widetilde{u}).P \,|\, Q\ ;\ R|\!\rangle_s^{n+1} \rightarrow \langle\!|z(\widetilde{u}).P \,|\, \phi(Q)\ ;\ R|\!\rangle_s^0$$

*and closed under $\equiv$, $(x)\_$, and the rules:*

$$\frac{P \rightarrow Q}{P \,|\, R \rightarrow Q \,|\, \phi(R)} \qquad \frac{P \rightarrow Q}{\langle\!|P\ ;\ R|\!\rangle_s^{n+1} \rightarrow \langle\!|Q\ ;\ R|\!\rangle_s^{n}}$$

$$\frac{P \rightarrow Q}{\langle\!|y(\widetilde{v}).R \,|\, R'\ ;\ P|\!\rangle_s^0 \rightarrow \langle\!|y(\widetilde{v}).R \,|\, \phi(R')\ ;\ Q|\!\rangle_s^0}$$

Rule (COM) is standard in process calculi and models the input-output interaction. Rule (FAIL) models workunits failure: when an abort is emitted, the corresponding workunit is terminated by setting the time stamp to 0, thus activating the event handler (last rule). On the contrary, aborts are not possible if the workunit is already terminated, namely every thread in the body has completed its own work. The inference rules lift reductions to parallel contexts and workunit contexts, updating them because a time slot is elapsed.

We say that $P$ has a barb $x$, and write $P \downarrow x$, if $P$ manifests an output on the free name $x$.

**Definition 3.** *Let $P \downarrow x$ be the least relation satisfying the rules and closed for $\equiv$:*

$$\begin{array}{rl}
\overline{x}\,\langle \widetilde{u}\rangle \downarrow x & \\
(z)P \downarrow x & \textit{if } P \downarrow x \textit{ and } x \neq z \\
(P \,|\, Q) \downarrow x & \textit{if } P \downarrow x \textit{ or } Q \downarrow x \\
\langle\!|P\ ;\ R|\!\rangle_s^0 \downarrow x & \textit{if } P \downarrow x \textit{ or } (\mathtt{inp}(P) \textit{ and } R \downarrow x) \\
\langle\!|P\ ;\ R|\!\rangle_s^{n+1} \downarrow x & \textit{if } P \downarrow x
\end{array}$$

## 4 The Calculus $\pi_t$

The advent of $\pi$-calculus has shown how diverse computational structures in both sequential and concurrent computing are uniformly representable as interacting processes. This allows the application of standard syntactic reasoning methods developed for process calculi to a wide variety of computational phenomena. However, in spite of its high expressive power and its interaction-based computing model, the $\pi$-calculus does not suffice for a complete and satisfactory description of basic elements of distributed computing systems. This is due to the difficult in decomposing some operations and phenomena in terms of message-passing, because they represent computational mechanisms left implicit or not treated in the $\pi$-calculus. For example, loss of message in transit, timers, process

failure and recovery are not taken into account by the $\pi$-calculus. The work by Berger — his PhD thesis [5] and other papers (for example [4])— is concerned on the study of an extension for the original $\pi$-calculus in order to provide a reasonable framework able to represent more realistic distributed systems. He tried to give extensions that can be basic and incremental, i.e. that combinations of a few simple extensions can represent a wide range of phenomena essentials to distributed systems.

### 4.1 Core Syntax

In this section we will illustrate the core syntax of the calculus presented by Berger.

$$
\begin{array}{lll}
P ::= & & (\textbf{processes}) \\
& \mathbf{0} & (\text{nil}) \\
& \mid \overline{x}\,\langle \widetilde{y} \rangle & (\text{message}) \\
& \mid x(\widetilde{y}).P & (\text{input}) \\
& \mid (x)P & (\text{restriction}) \\
& \mid P \mid P & (\text{parallel composition}) \\
& \mid\, !x(\widetilde{y}).P & (\text{lazy replication}) \\
& \mid \mathtt{timer}^n(x(\widetilde{v}).P, Q) & (\text{timer})
\end{array}
$$

A timer is a pair of processes, say $P$ and $Q$, and a deadline $n$, which represents the amount of disposed time. The semantic of timers is quite simple: a timer $\mathtt{timer}^n(\mu.P, Q)$ waits for a $\mu$ action until the total amount of time $n$ elapses. If the action $\mu$ is performed, the timer reduces to the continuation $P$, and the timeout continuation $Q$ is discarded. Contrariwise, if the time $n$ elapses without any action $\mu$, the timer reduces to the continuation $Q$, and the time-in continuation $P$ is discarded. The introduction of timers requires some extensions to the original $\pi$-calculus, which is not able to manage time. In particular, Berger and Honda introduce the *time-stepper function* $\phi_t$ , which indicates how the time passing influences the various constructs:

$$
\phi_t(P) \;=\; \begin{cases}
\mathtt{timer}^{n-1}(Q, R) & \text{if } P = \mathtt{timer}^n(Q, R),\ t > 1 \\
R & \text{if } P = \mathtt{timer}^n(Q, R),\ t \leqslant 1 \\
\phi_t(Q) \mid \phi_t(R) & \text{if } P = Q \mid R \\
(x)\phi_t(Q) & \text{if } P = (x)Q \\
P & \text{else}
\end{cases}
$$

Thus $\phi(P)_t$ ticks each timer in $P$ by one discrete degree: this can be thought of as the passing of, say, one second. Now we can introduce the reduction semantics, $\equiv_t$ is as usual.

**Definition 4.** *The* reduction relation $\rightarrow_t$ *is the least relation satisfying the following axioms and rules, and closed with respect to* $\equiv_t$ *and* $(x)\_$*:*

$$\overset{(\textsc{rep})}{\overline{x}\,\langle\widetilde{v}\rangle \mid !x(\widetilde{y}).P \;\rightarrow_t\; P\{\widetilde{v}/\widetilde{y}\} \mid !x(\widetilde{u}).P}$$

$$\overset{(\textsc{stop})}{\texttt{timer}^{n+1}(x(\widetilde{v}).P, Q) \mid \overline{x}\,\langle\widetilde{y}\rangle \;\rightarrow_t\; P\{\widetilde{y}/\widetilde{v}\}}$$

$$\overset{(\textsc{idle})}{P \rightarrow_t \phi_t(P)} \qquad \overset{(\textsc{par})}{\frac{P \rightarrow_t P'}{P \mid Q \rightarrow_t P' \mid \phi_t(Q)}}$$

Rules (STOP) and (PAR) are quite simple; they model the execution of timers and parallel processes. The rule (IDLE), instead, is a little more subtle: it allows the computation to pause or *idle* at arbitrary moments and, through repeated applications, for an unlimited period of time.

Now, we are ready to introduce the concept of *barb*, which will be used to prove the correctness of timers encoding. Informally, we say that $P$ has a barb $x$, and write $P \downarrow_t x$, if $P$ manifests an output on the free name $x$.

**Definition 5.** *Let $P \downarrow_t x$ be the least relation satisfying the rules and closed for $\equiv_t$:*

$$\begin{aligned} \overline{x}\,\langle\widetilde{y}\rangle &\downarrow_t x \\ (P \mid Q) &\downarrow_t x \text{ if } P \downarrow_t x \text{ or } Q \downarrow_t x \\ (x)P &\downarrow_t y \text{ if } P \downarrow_t y \text{ and } y \neq x \end{aligned}$$

## 5 Encoding Timers

In this section we show how to implement timers using workunits, then we will prove the correctness of this encoding. To this end we define the recursive function $[\![P]\!] : \pi_t \rightarrow \texttt{web}\pi$ which maps $\pi_t$ in $\texttt{web}\pi$ processes:

**Definition 6 ($\pi_t$ encoding in $\texttt{web}\pi$).** *Timers are defined by induction on $n$, for the missing cases it holds $[\![P]\!] = P$.*

$$[\![\texttt{timer}^1(y(\widetilde{u}).P, Q)]\!] = (x)(s)(\langle y(\widetilde{u}).\overline{x}\,\langle\widetilde{u}\rangle \;;\; [\![Q]\!]\rangle_s^1 \mid x(\widetilde{u}).[\![P]\!])$$

$$[\![\texttt{timer}^n(y(\widetilde{u}).P, Q)]\!] = (x)(s)(\langle y(\widetilde{u}).\overline{x}\,\langle\widetilde{u}\rangle \;;\; [\![\texttt{timer}^{n-1}(y(\widetilde{u}).P, Q)]\!]\rangle_s^1 \mid x(\widetilde{u}).[\![P]\!])$$

It is worth noting that this is not the only function satisfying our goals. We decided to adopt it after several investigations in order to achieve a tradeoff between mathematical elegance and quick understandability.

The encoding of a timer set to 1 behaves as follows: if the input-prefix $y(\widetilde{u})$ can react, the workunit emits the output message $\overline{x}\,\langle\widetilde{u}\rangle$, which triggers the continuation $P$. In this case, the workunit becomes the null process $\mathbf{0}$, and commits. Otherwise, if the input-prefix cannot react, it triggers the handler, reducing to the time-out continuation $Q$. It is easy to see that this is the expected behavior. The inductive case is quite similar, because it uses the same workunit set to 1. In this case, however, the event handling process is the recursive encoding of a timer, in which the deadline has approached of one unit. If the input-prefix

$y(\widetilde{u})$ can react, the workunit triggers the time-in continuation $P$, and commits; otherwise, the workunit runs the handler, and the timer encoding is called recursively.

The proposed encoding has obviously the required behavior, but it is *weak*, i.e. it requires an additional computational step, with respect to the native timer construct. In particular, when the input-prefix reacts, we must trigger the time-in continuation, while the timers in $\pi_t$ reduce directly. Let us illustrate this issue with an example. The $\pi_t$ program

$$\mathtt{timer}^n(x(\widetilde{u}).\overline{y}\,\langle\widetilde{z}\rangle, Q) \mid \overline{x}\,\langle\widetilde{v}\rangle \mid \mathtt{timer}^2(y(\widetilde{w}).R, S)$$

reduces, in one step, in the program

$$\overline{y}\,\langle\widetilde{z}\rangle \mid \mathtt{timer}^1(y(\widetilde{w}).R, S)$$

for the rules (STOP) and (PAR). Moreover, this evolves in $R\{\widetilde{z}/\widetilde{w}\}$. The correspondent encoding in $\mathtt{web}\pi$, instead, reduces, with an adjunctive $\tau$ step, in the program

$$\overline{y}\,\langle\widetilde{z}\rangle \mid (x)(s)(\langle\!| y(\widetilde{w}).\overline{x}\,\langle\widetilde{w}\rangle \;;\; [\![S]\!] |\!\rangle_s^0 \mid x(\widetilde{w}).[\![R]\!])$$

for (COM) and the rules for parallel and time elapsing. The point is that, while in the $\pi_t$ program the second timer had still a possibility to trigger, in its correspondent encoding the timer has elapsed and the time-out continuation is executed. Fortunately, this issue is harmless, because in $\pi_t$ we could execute idle steps, applying the rule (IDLE), in order to synchronize with the correspondent encoding in $\mathtt{web}\pi$. In particular, after triggering a timer, we execute an idle step:

$$\overline{y}\,\langle\widetilde{z}\rangle \mid \mathtt{timer}^1(y(\widetilde{w}).R, S)$$

reduces with the rule (IDLE) to

$$\overline{y}\,\langle\widetilde{z}\rangle \mid \mathtt{timer}^0(y(\widetilde{w}).R, S)$$

and finally to the time-out continuation $S$.

This example stress out an important difference between $\pi_t$ and $\mathtt{web}\pi$, i.e. the former is divergent, because it is possible to *idle* the computation for an unlimited period of time, while the latter does not allow to delay reductions to favor idle steps. So, what we are doing is encoding *one* of the many possible computations.

Now we will prove that a simulation exists between $\pi_t$ processes encodings and the processes themselves. Although it is possible to prove the existence of a simulation avoiding any particular constraints, for the sake of brevity we will show just a restricted proof. We will not allow a process $P$ in a parallel context $\mathsf{C}[\cdot]|P$ to be or to contain *time sensitive* operators (timers or timed workunits) and we use the notation $P^-$. As a consequence, we will avoid nested timed workunits because it is always possible to extrude by structural congruence and run them in parallel.

**Definition 7 (Contexts).** Process contexts, *noted* $\mathsf{C}[\cdot]$, *are defined by the following grammar:*

$$\mathsf{C}[\cdot] \ ::= [\cdot] \ \mid \ \mathsf{C}[\cdot]|P^{-} \ \mid \ (x)\mathsf{C}[\cdot]$$

*We always assume that when we write* $\mathsf{C}[P]$ *the resulting process is well-formed.*

The result can be easily extended to the general case but we got a longer proof. The basic idea behind the extended proof stands in the explanation above. When we introduce time sensitive operators in the context we have to force $\pi_t$ processes to synchronize with the correspondent encoding in $\texttt{web}\pi$ applying the rule (IDLE). The inductive case for parallel in the second part of the proof has to be extended with the relative sub-cases for time sensitive operators. This require some space and does not give additional hints about the result. For this reason we do not present here that part.

In order to present the proof we must introduce some preliminary definitions. If $\rightarrow$ is a binary relation, $\rightarrow_n$ is a shorthand for $\underbrace{\rightarrow \ldots \rightarrow}_{n}$. We write $\rightarrow^+$ if $\rightarrow_n$ for some $n > 0$. The Barbed Simulation is the basic machinery we use to provide the correctness proof:

**Definition 8 (Barbed Simulation).** *A barbed simulation* $\mathcal{S}$ *is a binary relation 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 similarity is the largest barbed simulation that is closed under contexts. P and Q are barbed similar and we write $P \lesssim Q$ if $P\,\mathcal{S}\,Q$ for some barbed simulation $\mathcal{S}$.

Since we are simulating processes over different systems we need a particular adaptation of the above definition:

**Proposition 1 (Barbed Simulation over Different Systems).** *Given two different systems* $(\mathbf{P}, \rightarrow_P, \downarrow_P, \equiv_P)$ *and* $(\mathbf{Q}, \rightarrow_Q, \downarrow_Q, \equiv_Q)$, *let us define* $\mathcal{S} = \{(P, Q) \mid P \in \mathbf{P}, Q \in \mathbf{Q}\}$ *such that* $(P, Q) \in \mathcal{S}$ *implies:*

1. *if* $P \downarrow_P x$ *then* $Q \downarrow_Q x$
2. *if* $P \rightarrow_P P'$ *and* $Q \rightarrow_Q^+ Q'$ *then* $P' \equiv_P P''$ *and* $(P'', Q'') \in \mathcal{S}$ *with* $Q'' \equiv_Q Q'$

In the following we consider the two systems $(\mathbf{P}, \rightarrow, \downarrow, \equiv)$ and $(\mathbf{Q}, \rightarrow_t, \downarrow_t, \equiv_t)$ where $\mathbf{P}$ are $\texttt{web}\pi$ processes and $\mathbf{Q}$ are $\pi_t$ processes. The following theorem proves that, if a timer encoded by the timed workunit behaves in a certain way, also $\pi_t$ timers can behave in the same way.

**Theorem 1 (Barbed Similarity between $[\![P]\!]$ and $P$).**

$$\forall\, P \ \in \pi_t,\ \mathsf{C}[[\![P]\!]] \lesssim \mathsf{C}[P]$$

*Proof.* The relation $\mathcal{S}$ defined as follows is a barbed simulation:

$$\begin{aligned}\mathcal{S} &= \{(\llbracket P \rrbracket, P) \mid P \in \pi_t\} \\ &\cup \{((x)(s)(\langle\!| \overline{x}\,\langle \widetilde{v}\rangle \;;\; \llbracket Q \rrbracket |\!\rangle_s^0 \mid x(\widetilde{u}).\llbracket P \rrbracket), P\{\widetilde{v}/\widetilde{u}\}) \mid P, Q \in \pi_t\} \\ &\cup \{((x)(s)(\langle\!| y(\widetilde{u}).\overline{x}\,\langle \widetilde{v}\rangle \;;\; \llbracket Q \rrbracket |\!\rangle_s^0 \mid x(\widetilde{u}).\llbracket P \rrbracket), Q) \mid P, Q \in \pi_t\} \\ &\cup \equiv_t\end{aligned}$$

Let us prove the two conditions required to have a simulation.

1. Firstly, if $\mathsf{C}[\llbracket P \rrbracket] \downarrow x$ then $\mathsf{C}[P] \downarrow_t x$. By induction over contexts:
   (a) **Base Case:** if $\mathsf{C}[\cdot]$ is $[\cdot]$: By induction on the structure of $P$:
      i. $P$ is not a timer: the statement is obvious, because the encoding in this case is the identity function;
      ii. $P$ is a timer: its encoding does not show any barb, so the statement is banally true;
   (b) **Inductive Case for Restriction**: we have to prove that if $(y)\mathsf{C}[\llbracket P \rrbracket] \downarrow x$ then $(y)\mathsf{C}[P] \downarrow_t x$. If $\mathsf{C}[\llbracket P \rrbracket] \downarrow x$, there are two possible cases:
      i. we restrict the actual name $x$: $(x)\mathsf{C}[\llbracket P \rrbracket] \downarrow$, so the statement is banally true.
      ii. we restrict the name $y$, $y \neq x$: if $(y)\mathsf{C}[\llbracket P \rrbracket] \downarrow x$, $(y)\mathsf{C}[P] \downarrow_t x$, so the statement is true.
   (c) **Inductive Case for Parallel**: we have to prove that if $\mathsf{C}[\llbracket P \rrbracket] \,|\, Q \downarrow x$ then $\mathsf{C}[P] \,|\, Q \downarrow_t x$:
   if $\mathsf{C}[\llbracket P \rrbracket] \,|\, Q \downarrow x$, then $\mathsf{C}[\llbracket P \rrbracket] \downarrow x$ or $Q \downarrow x$; moreover, $\mathsf{C}[P] \downarrow_t x$ or $Q \downarrow_t x$ by inductive hypothesis. This means that $\mathsf{C}[P] \,|\, Q \downarrow_t x$.

2. The second part of the proof consists in showing that if $\mathsf{C}[\llbracket P \rrbracket] \rightarrow P'$ then $\mathsf{C}[P] \rightarrow_t^+ P''$ and $P' \,\mathcal{S}\, P''$. By induction over contexts:
   (a) **Base Case:** if $\mathsf{C}[\cdot]$ is $[\cdot]$: By induction on the structure of $P$:
      i. $P$ is not a timer: the encoding of $P$ is the identity function, and this preserves the relation:

$$\llbracket P \rrbracket = P \text{ and } \llbracket P \rrbracket \rightarrow P', \text{ then } \exists\, P'' \text{ such that } P \rightarrow_t P'' \text{ and } P' = P''$$

      ii. $P$ is a timer of the shape $\mathtt{timer}^1(y(\widetilde{u}).A, B)$:

$$\llbracket \mathtt{timer}^1(y(\widetilde{u}).A, B) \rrbracket = (x)(s)(\langle\!| y(\widetilde{u}).\overline{x}\,\langle \widetilde{u}\rangle \;;\; \llbracket B \rrbracket |\!\rangle_s^1 \mid x(\widetilde{u}).\llbracket A \rrbracket)$$

      This object cannot reduce by itself, it would require some other process running in parallel to trigger $y(\widetilde{u})$ or to make possible for the time to pass. Thus, the statement is obviously true.

      iii. $P$ is a timer of the shape $\mathtt{timer}^n(y(\widetilde{u}).A, B)$:

$$\llbracket \mathtt{timer}^n(y(\widetilde{u}).A, B) \rrbracket = (x)(s)(\langle\!| y(\widetilde{u}).\overline{x}\,\langle \widetilde{u}\rangle \;;\; \llbracket \mathtt{timer}^{n-1}(y(\widetilde{u}).A, B) \rrbracket |\!\rangle_s^1 \mid x(\widetilde{u}).\llbracket A \rrbracket)$$

      this object cannot reduce. The same considerations of above holds.

(b) **Inductive case for restriction:** if $\mathsf{C}[\cdot]$ is $(x)\mathsf{C}[\cdot]$, we have to prove that if $(x)\mathsf{C}[[\![P]\!]] \to P'$, then $(x)\mathsf{C}[P] \to_t P''$ and $P' \,\mathcal{S}\, P''$. If $(x)\mathsf{C}[[\![P]\!]] \to (x)Q$, then $\mathsf{C}[[\![P]\!]] \to Q$. By inductive hypothesis, $\mathsf{C}[P] \to_t Q'$ such that $Q \,\mathcal{S}\, Q'$. By structural congruence, $(x)Q \,\mathcal{S}\, (x)Q'$.

(c) **Inductive case for parallel:** if $\mathsf{C}[\cdot]$ is $\mathsf{C}[\cdot]|P^-$, we have to prove that if $\mathsf{C}[[\![P]\!]] \,|\, Q \to P'$, then $\mathsf{C}[P] \,|\, Q \to_t P''$ and $P' \,\mathcal{S}\, P''$. $\mathsf{C}[[\![P]\!]] \,|\, Q$ can reduce for three reasons:

i. $\mathsf{C}[[\![P]\!]] \,|\, Q \to \mathsf{C}[[\![P]\!]]' \,|\, Q$: we can say that $\mathsf{C}[[\![P]\!]] \to \mathsf{C}[[\![P]\!]]'$ and, applying the inductive hypothesis, $\mathsf{C}[P] \to_t \mathsf{C}[P]'$ and $\mathsf{C}[[\![P]\!]]' \,\mathcal{S}\, \mathsf{C}[P]'$. Now, $[\![\mathsf{C}[P]]\!]' \,|\, Q \,\mathcal{S}\, \mathsf{C}[P]' \,|\, Q$.

ii. $\mathsf{C}[[\![P]\!]] \,|\, Q \to \mathsf{C}[[\![P]\!]] \,|\, Q'$: This case is symmetric with respect to the previous one.

iii. $\mathsf{C}[[\![P]\!]] \,|\, Q \to \mathsf{C}[[\![P]\!]]' \,|\, Q'$: since $Q$ is not time sensitive, we must consider only the following sub-cases:

A. $P$ does not contain a timer: $[\![\cdot]\!]$ is the identity function, and the statement is obvious.

B. $Q$ triggers a timer in $P$ receiving the message $\overline{y}\,\langle\widetilde{v}\rangle$ :
the process has the shape $[\![\mathtt{timer}^n(y(\widetilde{u}).A, B)]\!] \,|\, \overline{y}\,\langle\widetilde{v}\rangle \,|\, Q''$ and reduces to $(x)(s)(\{\!|\overline{x}\,\langle\widetilde{v}\rangle \,;\, [\![\mathtt{timer}^{n-1}(y(\widetilde{u}).A, B)]\!]\}\!|^0_s \,|\, x(\widetilde{u}).[\![A]\!]) \,|\, Q''$.
On the other side, $\mathtt{timer}^n(y(\widetilde{u}).A, B) \,|\, \overline{y}\,\langle\widetilde{v}\rangle \,|\, Q''$ in one step reduces to $A\{{}^{\widetilde{v}}/_{\widetilde{u}}\} \,|\, Q''$.
Now, $(x)(s)(\{\!|\overline{x}\,\langle\widetilde{v}\rangle \,;\, [\![\mathtt{timer}^{n-1}(y(\widetilde{u}).A, B)]\!]\}\!|^0_s \,|\, x(\widetilde{u}).[\![A]\!]) \,|\, Q''$ is in relation $\mathcal{S}$ with $A\{{}^{\widetilde{v}}/_{\widetilde{u}}\} \,|\, Q''$ by definition of $\mathcal{S}$ and inductive hypothesis.

C. $P$ contains a timer and $Q$ makes it possible for the time to pass and for the workunit to trigger the handler. The timer can have two possible forms:

– $P$ is a timer of the shape $\mathtt{timer}^1(y(\widetilde{u}).A, B)$:
$[\![\mathtt{timer}^1(y(\widetilde{u}).A, B)]\!] = (x)(s)(\{\!|y(\widetilde{u}).\overline{x}\,\langle\widetilde{u}\rangle \,;\, [\![B]\!]\}\!|^1_s \,|\, x(\widetilde{u}).[\![A]\!])$
This process reduces to $(x)(s)(\{\!|y(\widetilde{u}).\overline{x}\,\langle\widetilde{v}\rangle \,;\, [\![B]\!]\}\!|^0_s \,|\, x(\widetilde{u}).[\![A]\!])$.
On the other hand, $\mathtt{timer}^1(y(\widetilde{u}).A, B) \to_t B$ and, by definition of $\mathcal{S}$: $(x)(s)(\{\!|y(\widetilde{u}).\overline{x}\,\langle\widetilde{v}\rangle \,;\, [\![B]\!]\}\!|^0_s \,|\, x(\widetilde{u}).[\![A]\!]) \,\mathcal{S}\, B$.

– $P$ is a timer of the shape $\mathtt{timer}^n(y(\widetilde{u}).A, B)$:

$[\![\mathtt{timer}^n(y(\widetilde{u}).A, B)]\!] = (x)(s)(\{\!|y(\widetilde{u}).\overline{x}\,\langle\widetilde{u}\rangle \,;\, [\![\mathtt{timer}^{n-1}(y(\widetilde{u}).A, B)]\!]\}\!|^1_s \,|\, x(\widetilde{u}).[\![A]\!])$

This process reduces to:
$(x)(s)(\{\!|y(\widetilde{u}).\overline{x}\,\langle\widetilde{v}\rangle \,;\, [\![\mathtt{timer}^{n-1}(y(\widetilde{u}).A, B)]\!]\}\!|^0_s \,|\, x(\widetilde{u}).[\![A]\!])$.
On the other hand, $\mathtt{timer}^n(y(\widetilde{u}).A, B) \to_t \mathtt{timer}^{n-1}(y(\widetilde{u}).A, B)$ by the rule (IDLE) and, by definition of $\mathcal{S}$:

$(x)(s)(\{\!|y(\widetilde{u}).\overline{x}\,\langle\widetilde{v}\rangle \,;\, [\![\mathtt{timer}^{n-1}(y(\widetilde{u}).A, B)]\!]\}\!|^0_s \,|\, x(\widetilde{u}).[\![A]\!]) \,\mathcal{S}\, \mathtt{timer}^{n-1}(y(\widetilde{u}).A, B)$.

## 6 Conclusions

In this paper we showed how $\texttt{web}\pi$ is able to cope with timing issues in the same way as Berger-Honda did. To show this we encoded their time construct, called timer, in our timed workunit and we proved a simulation theorem. The proof has some limitations. Firstly, we did not show the proof including time sensitive operators in parallel. As explained this was just for the sake of brevity and we gave some hints about the extension of the proof. A second point, instead, deserves more attention. Unfortunately, the result we presented is not symmetric, in the sense that we proved simply a simulation and not a bisimulation. For this reason our result could be considered too limited. However, we are working on extended results and we are quite confident about related theorems. We strongly rely on the fact that an analogous result can be proved for the viceversa. In particular $\mathsf{C}[P] \lesssim \mathsf{C}[[\![P]\!] \,|\, \tau^*]$ should hold. Another interesting result to verify is whether $P \lesssim Q \Leftrightarrow [\![P]\!] \lesssim [\![Q]\!]$. Presently, we are also showing how timed constructs of composition languages (e.g. BPEL alarm clocks) can be formalized in this timed calculus. This work can be intended as an extension of we did in [23] for the untimed ones but presenting here this kind of formalization goes beyond the scope of the paper.

Finally, we want to remark the fact that any mathematical rigor becomes pointless without the ability to provide conceptual tools for analysis and reasoning or software tools for verification. In this sense contracts conformance verification between different services should be investigated in the future.

## References


1. Business Process Execution Language open issues list. http://www.oasis-open.org/apps/group_public/download.php/11285/wsbpel_issues34.html.
2. Questions on Choreology's coordinated choreographies proposals. http://lists.w3.org/Archives/Public/public-ws-chor/2004Nov/0016.html.
3. Assaf Arkin et al. *Web Service Business Process Execution Language*. OASIS, February 2005.
4. Martin Berger. *Towards Abstractions for Distributed Systems*. PhD thesis, Imperial College, London, 2002.
5. Martin Berger and Kohei Honda. The Two-Phase Commit Protocol in an Extended $\pi$-Calculus. In *Proc. EXPRESS'00*, volume 39 of *ENTCS*, 2000.
6. Laura Bocchi, Cosimo Laneve and Gianluigi Zavattaro. A calculus for long running transactions. In *Proc. FMOODS '03*, volume 2884 of *LNCS*. Springer-Verlag, 2003.
7. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte and D. Winer. Simple Object Access Protocol (SOAP) 1.1. [http://www.w3.org/TR/SOAP/], W3C, Note 08, May 2000.
8. Roberto Bruni, Cosimo Laneve and Ugo Montanari. Orchestrating transactions in the join calculus. In *CONCUR 2002*, volume 2421 of *LNCS*. Springer-Verlag, 2002.
9. Roberto Bruni, Hernán Melgratti and Ugo Montanari. Theoretical foundations for compensations in flow composition languages. In *POPL*. ACM, 2005. To appear.
10. Michael Butler and Carla Ferreira. An operational semantics for stac, a langage for modelling long-running businness transactions. In *COORDINATION 2004*, volume 2949 of *LNCS*. Springer-Verlag, 2004.

11. Erik Christensen, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana. *Web Services Description Language (WSDL 1.1)*. W3C, 2001.
12. World Wide Web Consortium. Extensible Markup Language (XML) 1.0. W3C Recommendation, 1998. http://www.w3.org/TR/REC-XML.
13. Philippa Gardner, Cosimo Laneve and Lucian Wischik. The fusion machine (extended abstract). In *CONCUR 2002*, volume 2421 of *LNCS*. Springer-Verlag, 2002.
14. Claudio Guidi, Roberto Lucchi and Manuel Mazzara. A formal framework for web services coordination. In *FOCLASA 2004*. To appear in ENTCS, Elsevier. To appear.
15. Tony Hoare. Long-running transactions. http://research.microsoft.com.
16. Iain Houston, Mark C. Little, Ian Robinson, Santosh K. Shrivastava and Stuart M. Wheater. The CORBA activity service framework for supporting extended transactions. Softw. Pract. Exper. 33(4): 351-373 (2003).
17. Nickolas Kavantzas. Aggregating web services: Choreography and ws-cdl. http://lists.w3.org/Archives/Public/www-archive/2004Jun/att-0008/WS-CDL-April200 4.pdf.
18. Nickolas Kavantzas, David Burdett, Gregory Ritzinger and Yves Lafon. *Web Services Choreography Description Language Version 1.0*. OASIS, October 2004.
19. Cosimo Laneve and Gianluigi Zavattaro. Foundations of web transactions. In *Fossacs'05*, volume 3441 of *LNCS*. Springer-Verlag, 2005.
20. Cosimo Laneve and Gianluigi Zavattaro. Web$\pi$ at work. In *In Proc. of Symposium on Trustworthy Global Computing (TGC'05)*, LNCS. Springer-Verlag, 2005. To appear.
21. Frank Leymann. Web Services Flow Language (WSFL 1.0). [http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf], Member IBM Academy of Technology, IBM Software Group, 2001.
22. Mark Little. Web services transactions: Past, present and future. http://www.idealliance.org/papers/dx_xml03/ html/abstract/05-02-02.html.
23. Roberto Lucchi and Manuel Mazzara. A $\pi$-calculus based semantics for ws-bpel. *Journal of Logic and Algebraic Programming (JLAP)*. To appear.
24. Manuel Mazzara and Sergio Govoni. A case study of web services orchestration. In *COORDINATION 2005*, volume 3454 of *LNCS*. Springer-Verlag, 2005.
25. Manuel Mazzara and Roberto Lucchi. A framework for generic error handling in business processes. In *First International Workshop on Web Services and Formal Methods (WS-FM)*, volume 105 of *ENTCS*. Elsevier, 2004.
26. Robin Milner. *Communicating and Mobile Systems: the $\pi$-Calculus*. Cambridge University Press, 1999.
27. Robin Milner, Joachim Parrow and David Walker. A Calculus for Mobile Processes. *Journal of Information and Computation*, 100:1–77, 1992.
28. Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003.
29. S. Thatte. XLANG: Web Services for Business Process Design. Microsoft Corporation, 2001.
30. Wil van der Aalst. Pi-calculus versus petri nets: Let us eat 'humble pie' rather than further inflate the.
31. W3C. *HTTP - HyperText Transfer Protocol Specification*. www.w3.org/protocols.