# Towards Abstractions for Web Services Composition

Manuel Mazzara

# Agenda

# Crossing Organizational Boundaries

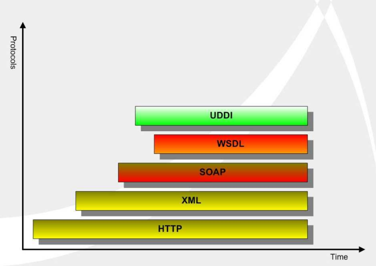
With E-business applications processes need to cross organizational boundaries

# Web Services

Web Services are a set of technologies which promise to facilitate B2B integration using a standard web-messaging infrastructure

- supports Service Oriented Computing

No revolution about Web Services, simply an evolution based on already existing Internet protocols



# Service Oriented Computing

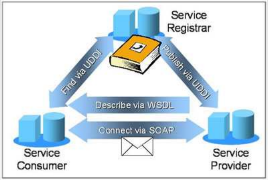
- an emerging paradigm for distributed computing and e-business processing
- finds its origin in object-oriented and component computing

Goal: enabling developers to build networks of integrated and collaborative applications, regardless of both the platform where the application or service runs and the programming language used to develop them.

# Service Oriented Architecture

The service-oriented architecture is the latest of a long series of attempts in software engineering addressing the reuse of software components

- function and the concept of API
- object (classes, inheritance, polymorphism...)
- service (consumer, provider, registrar...)



# Web Services Composition

- In the Web Services Programming Model a service can itself use several other services and each of these services will be based on the same model
- It is a recursive use of the model transparent to the final consumer
- Web services technologies provide a mechanism to build complex services out of simpler ones, i.e. Web services Composition

# State of the Art

Different organizations are presently working on additional stack layers which have to deal with the new approach of composing Web services on a workflow base for business automation purposes

- IBMs WSFL
- Microsofts XLANG
- WS-BPEL

WS-BPEL aims at integrating both WSFL and XLANG

# WS-BPEL

- It is a workflow-based programming language that describes sophisticated business processes that orchestrate Web services
- It allows for a mixture of block and graph-structured process models (the language is expressive at the price of being complex)
- BPEL represents the most credited candidate to become a future standard in the field of Web services composition

# Business Process

A business process specifies the potential execution order of operations originating from a collection of Web Services

- the shared data passed between these services
- the trading partners that are involved in the joint process
- their roles with respect to the process
- joint exception handling condition

# Requirements for Workflow-based Composition (1)

- Flexibility: a clear separation between the process logic and the Web services invoked. Got trough an orchestration engine
- Basic and structured activities: support activities for both communicating with other Web services and handling workflow semantics
- Recursive composition: a business process can itself be exposed as a Web service, enabling business processes to be aggregated to form higher level processes

# Requirements for Workflow-based Composition (2)

- Persistence and correlation: a mechanism to manage data persistence and correlate requests in order to build higher-level conversations
- Exception handling and transactions: services that are long-running must also manage exceptions and transactional integrity

# Web Services and Business Transactions (1)

Web Services environment requires that several Web Service operations have transactional properties and be treated as a single logical unit of work. Example:

- a manufacturer develops a Web Service-based solutions to automate order and delivery with its suppliers
- The transaction between the manufacturer and its suppliers can only be considered successful once all parts are delivered to their final destination
- This could be days or weeks after the placement of the order

# Web Services and Business Transactions (2)

The ACID (atomic, consistent, durable, isolated) model involves transactions that are

- tightly coupled
- occur between trusted systems
- involve short periods of time

It is not suitable for loosely coupled environments such as Web Service-based business. The major issue is the isolation of a database transaction. This property requires resource locking that is impractical in the business world

# Long Running Transactions

We refer to nonACID transactions as Long Running

- Compensation

It is an application-specific activity which attempts to reverse the effects of a previous activity carried out as part of a larger unit of work which is being abandoned

- it is itself a part of the business logic and must be explicitly designed

# Computational Model

In the early years of digital computers almost all machines were based purely on the Von Neumann architecture

- Turing Machines
- $\lambda$-calculus

Nowadays most application areas of computing involve interactions and systems in which many components are concurrently active - furthermore there have been the advent of mobile computing

- Process Algebras

# Process Algebras

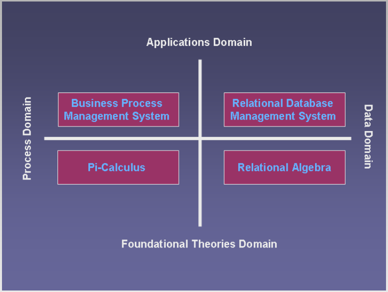
It is a means of algebraically specifying the behavior of concurrent and distributed computational processes.
The $\pi$-calculus:

- was developed in the late 80s to be a theory of mobile systems
- it concerns computations in which the communication topology is dynamic
- a practical well-known application is represented by the XLANG Scheduler in Microsoft BizTalk Server 2000

# Main Goal

Formally addressing the problem of Web Services Composition

- particular attention to Error Handling



# Composition and the $\pi$-calculus

Languages like XLANG, WS-BPEL and WS-CDL are claimed to be based on the $\pi$-calculus to allow rigorous mathematical reasoning. Despite all this hype, strong relations between theory and practice are not always evident:

- few conceptual instruments for analysis and reasoning
- few software verification techniques and tools

Without the ability to show a great practical impact, mathematical rigor risks to become pointless.

# Contributions (1)

- an analysis of the relationships between process algebras (in particular the $\pi$-calculus) and workflow management technologies in the context of web services composition
- a proposal of a calculus for orchestration: $\mathtt{web}\pi_\infty$
- the theory of this calculus
- representation of realistic e-commerce transactional scenarios guaranteeing consistency properties (despite of the unsuitability of ACIDity)

## Contributions (2)

- the WS-BPEL formal semantics in term of this calculus
- a simplification for the WS-BPEL Recovery Framework: unification of the mechanisms (fault, compensation and event handling)
- some insight about compiling optimizations and compilers alternative design choices for orchestration engines
- mathematical comparison between an extended timed version of $\mathtt{web}\pi_\infty$ ($\mathtt{web}\pi$ by Laneve-Zavattaro) and past models of time in process algebra (Berger-Honda)

# Syntax

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

# Structural congruence

1. Scope laws:

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

2. Workunit laws:

$$
\begin{gathered}
\langle\!|\mathbf{0} \ ;\ Q|\!\rangle_x \equiv \mathbf{0} \\
\langle\!|\langle\!|P \ ;\ Q|\!\rangle_y \mid R \ ;\ R'|\!\rangle_x \equiv \langle\!|P \ ;\ Q|\!\rangle_y \mid \langle\!|R \ ;\ R'|\!\rangle_x
\end{gathered}
$$

3. Floating law:

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

# Reduction relation

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

## Definition

The *reduction relation* $\rightarrow$ is the least relation satisfying the following axioms and rules, and closed with respect to $\equiv$, $(x)\_$ , $\_ \mid \_$, and $\langle\!| \_ \,;\, R |\!\rangle_z$:

(COM)

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

(REP)

$$\overline{x}\,\widetilde{v} \mid !x(\widetilde{u}).P \quad\rightarrow\quad P\{\widetilde{v}/\widetilde{u}\} \mid !x(\widetilde{u}).P$$

(FAIL)

$$\overline{x} \mid \langle\!| \textstyle\prod_{i\in I} \sum_{s\in S} x_{is}(\widetilde{u_{is}}).P_{is} \,;\, Q |\!\rangle_x \quad\rightarrow\quad \langle\!| Q \,;\, \mathbf{0} |\!\rangle \qquad (I \neq \emptyset)$$

# Extensional Semantics (1)

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

**Definition**

Let $P \downarrow x$ be the least relation satisfying the rules:

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

# Extensional Semantics (2)

## Definition

A barbed bisimulation $\mathcal{S}$ is a symmetric 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 bisimilarity*, denoted with $\approx$, is the largest barbed bisimulation that is also a congruence.

# The Labeled Semantics

Only few rules...

## Definition

The *transition relation* of $\mathtt{web}\pi_\infty$ processes, noted $\xrightarrow{\mu}$, is the least relation satisfying the rules:

$$(\text{ABORT})\ \frac{\mathsf{inp}(P)}{\langle\!|P\ ;\ Q|\!\rangle_x \xrightarrow{x()} \langle\!|\mathsf{xtr}(P) \mid Q\ ;\ \mathbf{0}|\!\rangle}$$

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

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

# Remark 1

We preserve the workunit structure after its abort to have the input predicate falsity stable with respect to the transition relation:

- we do not want that having $\neg \mathrm{inp}(P)$ and $P \xrightarrow{\mu} P'$ then $\mathrm{inp}(P)$
- the opposite makes sense, i.e. if $\mathrm{inp}(P)$ and $P \xrightarrow{\mu} P'$ then $\neg \mathrm{inp}(P)$ (for example in $\overline{x} \mid x().\mathbf{0}$)

# Remark 2

The side condition inp($P$) in the rule (self) should be written inp($Q$) referring to the pending state of some input in the process Q *after* the $x$ signal.
It is safe to write inp($P$) instead of inp($Q$). because of the following proposition:
Let $P$ be a $\mathtt{web}\pi_\infty$ process:

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

**Proof.**

The proof is by induction on the structure of $P$ □

# Asynchronous Bisimulation

## Definition

*Asynchronous bisimulation* is the largest symmetric binary relation $\dot{\approx}_a$ such that $P \dot{\approx}_a Q$ implies:

1. if $P \xrightarrow{\tau} P'$, then $Q \stackrel{\tau}{\Longrightarrow} Q'$ and $P' \dot{\approx}_a Q'$;
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 $P' \dot{\approx}_a Q'$;
3. if $P \xrightarrow{x(\widetilde{u})} P'$ then
   1. either $Q \stackrel{x(\widetilde{u})}{\Longrightarrow} Q'$, and $P' \dot{\approx}_a Q'$;
   2. or $Q \stackrel{\tau}{\Longrightarrow} Q'$, and $P' \dot{\approx}_a (Q' | \overline{x}\,\widetilde{u})$.

## Counterexample for congruence

$\dot{\approx}_a$ is not a congruence

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

$P \dot{\approx}_a Q$ (they both cannot move), inp($Q$) holds but inp($P$) not
Consider the context $\langle\!|\, C_\pi[\cdot] \; ; \; \overline{y} \,|\!\rangle_x$ and the rules (self) and (abort) you can see that the processes

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

behave differently with respect to the asynchronous bisimulation

# Still not a Congruence

## Definition

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

Another counterexample:

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

$P \stackrel{\bullet}{\approx}_a Q$, $\mathrm{inp}(P) = \mathrm{inp}(Q)$ but $\mathrm{xtr}(P) \neq \mathrm{xtr}(Q)$ because $\mathrm{xtr}(P) = \langle\!| z().u() \;;\; \mathbf{0} |\!\rangle$ and $\mathrm{xtr}(Q) = \langle\!| x().y() \;;\; \mathbf{0} |\!\rangle$.

# Congruence

## Definition

A binary relation $\mathcal{R}$ over processes is extract-closed if $P \mathcal{R} Q$ implies $\mathrm{xtr}(P) = \mathrm{xtr}(Q)$.

## Definition

Labeled bisimilarity $\approx_a$ is the greatest asynchronous bisimulation contained into $\dot{\approx}_a$ that is input predicate-closed and extract-closed.

## Theorem

*$\approx_a$ is a congruence, i.e. given two processes $P$ and $Q$ such as $P \approx_a Q$ then $C_\pi[P] \approx_a C_\pi[Q]$.*

## Proof.

By induction over contexts □

# Auxiliary Lemmas (1)

## Lemma

*Let $P$ be a $\texttt{web}\pi_\infty$ process. Then the followings hold:*

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

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

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

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

# Auxiliary Lemmas (2)

## Lemma

*Let $P$ be a* $\mathtt{web}\pi_\infty$ *process. Then the followings hold:*

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

# Auxiliary Lemmas (3)

## Lemma

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

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

# The Theorem

## Theorem

*$P \approx_a Q$ implies $\approx$.*

## Proof.

The previous lemma proved that $\approx_a$ is a barbed bisimulation. We have also proved that $\approx_a$ is a congruence. Because $\approx$ is the largest one by definition the statement follows. $\square$

# Relevant Examples

Handlers Reducibility

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

Decoupling of Service and Recovery Logics

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

# Case Study

I presented an implementation in $\mathtt{web}\pi_\infty$ of a classical e-business scenario (customer, provider, web portal)

1. It is not a toy application, the logic of real transactional is not very different from this one
2. The complete logic of the application could be more complicated but the single transactional part is often limited for many reason (e.g. to satisfy performance requirements of real system having to cope with an huge amount of incoming messages)

# The Semantics of BPEL (1)

1. BPEL is not equipped with formal semantics
2. it includes a large number of aspects
3. it is difficult to formally reason on processes behavior

# The Semantics of BPEL (2)

1. the semantics of a significant BPEL fragment is formally addressed
2. particular attention for the specification of event, fault and compensation handlers behavior
3. I advocate that three different mechanisms for error handling are not necessary
4. $\texttt{web}\pi_\infty$ is based on the idea of event notification as the unique error handling (similar ideas in the extensions of the CORBA transactional system or Java Transactional Web Services (JTWS)).

# Core BPEL

$$
\begin{array}{lll}
A ::= & & \\
& \texttt{empty} & \textbf{(empty)} \\
\mid & \texttt{invoke}(x_s, \tilde{i}, \tilde{o}) & \textbf{(synch invoke)} \\
\mid & \texttt{invoke}(x_s, \tilde{i}) & \textbf{(asynch invoke)} \\
\mid & \texttt{receive}(x_s, \tilde{i}) & \textbf{(receive)} \\
\mid & \texttt{reply}(x_s, \tilde{o}) & \textbf{(reply)} \\
\mid & \texttt{throw}(f, \tilde{o}) & \textbf{(throw)} \\
\mid & \texttt{compensate}(z, \tilde{o}) & \textbf{(compensate)} \\
\mid & \texttt{sequence}\ (A, A) & \textbf{(sequence)} \\
\mid & \texttt{flow}\ (A, A) & \textbf{(parallel)} \\
\mid & \texttt{pick}\ ((x, \tilde{i}_1, A), (x, \tilde{i}_2, A)) & \textbf{(alternative)} \\
\mid & \texttt{scope}_z(A, S_e, S_f, A) & \textbf{(scope)}
\end{array}
$$

## Basic and Structured Activities

$$
\begin{aligned}
&[\![\texttt{empty}]\!]_{\overline{y}\,\tilde{u}} = \overline{y}\,\tilde{u}\\
&[\![\texttt{invoke}(x_s, \tilde{i})]\!]_{\overline{y}\,\tilde{u}} = \overline{x_s}\,\tilde{i} \mid \overline{y}\,\tilde{u}\\
&[\![\texttt{invoke}(x_s, \tilde{i}, \tilde{o})]\!]_{\overline{y}\,\tilde{u}} = (r)(\overline{x_s}\,r, \tilde{i} \mid r(\tilde{o}).\overline{y}\,\tilde{u})\\
&[\![\texttt{receive}(x_s, \tilde{i})]\!]_{\overline{y}\,\tilde{u}} = x_s(r, \tilde{i}).\overline{y}\,\tilde{u}\\
&[\![\texttt{reply}(x_s, \tilde{o})]\!]_{\overline{y}\,\tilde{u}} = \overline{x_s}\,\tilde{o} \mid \overline{y}\,\tilde{u}\\
&[\![\texttt{throw}(f, \tilde{o})]\!]_{\overline{y}\,\tilde{u}} = (r)(\ \overline{throw} \mid \overline{f}\,r, \tilde{o} \mid r().\overline{y}\,\tilde{u})\\
&[\![\texttt{compensate}(z, \tilde{o})]\!]_{\overline{y}\,\tilde{u}} = \overline{z}\,\tilde{o} \mid \overline{y}\,\tilde{u}\\
&[\![\texttt{sequence}\ (A', A'')]\!]_{\overline{y}\,\tilde{u}} = (y')([\![A']\!]_{\overline{y'}\,\tilde{v}} \mid y'(\tilde{u}).[\![A'']\!]_{\overline{y}\,\tilde{u}}) \qquad \tilde{v} = \mathrm{fn}(A'')\\
&[\![\texttt{flow}\ (A', A'')]\!]_{\overline{y}\,\tilde{u}} = (y')(y'')([\![A']\!]_{\overline{y'}\,\tilde{u'}} \mid [\![A'']\!]_{\overline{y''}\,\tilde{u''}} \mid y'(\tilde{u'}).y''(\tilde{u''}).\overline{y}\,\tilde{u})\\
&[\![\texttt{pick}\ \ ((x_1, \tilde{i_1}, A), (x_2, \tilde{i_2}, A))\ ]\!]_{\overline{y}\,\tilde{u}} = x_1(\tilde{i_1}).[\![A']\!]_{\overline{y}\,\tilde{u}} + x_2(\tilde{i_2}).[\![A'']\!]_{\overline{y}\,\tilde{u}}
\end{aligned}
$$

# Encoding scopes in $\mathtt{web}\pi_\infty$

Let us present for brevity the case of the Event handler

$$\left(\begin{array}{lcl} EH(S_e, y_{eh}) & = & (y')(\{e_x \mid x \in h_e(S_e)\}) \\ & & en_{eh}().(\langle\!|\, \prod_{(x,\widetilde{u},A)\in S_e} \ !\ x(\widetilde{u}).\overline{e_x}\,\widetilde{u}\ ;\ \overline{y_{eh}}\,|\!\rangle_{dis_{eh}} \\ & & \mid \prod_{(x,\widetilde{u},A_x)\in S_e} \ !\ e_x(\widetilde{u}).[\![A_x]\!]_{\overline{y'}}\ ) \end{array}\right)$$

Recalling a theorem proved:

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

we can write

$$\left(\begin{array}{lcl} EH(S_e, y_{eh}) & = & (y')(e_x) \\ & & en_{eh}().((y'')(\langle\!|\, !\ x(\widetilde{u}).\overline{e_x}\,\widetilde{u}\ ;\ \overline{y''}\,|\!\rangle_{dis_{eh}} \\ & & \mid \langle\!|\,(w)\ w(\widetilde{u})\ ;\ \overline{y_{eh}}\,|\!\rangle_{y''}) \\ & & \mid !\ e_x(\widetilde{u}).[\![A_x]\!]_{\overline{y'}}\ ) \end{array}\right)$$

## Consequences

1. We are able to separate always the body and the recovery logics of a workunit expressing the event handler behavior
2. This in general is possible not only when the recovery logic is a simple output (as in this case) but in all the other cases, on the basis of another theorem:
$\langle\!| P \; ; \; Q |\!\rangle_x \approx_a (x')(\langle\!| P \; ; \; \overline{x'} |\!\rangle_x \mid \langle\!| x'().Q \; ; \; \mathbf{0} |\!\rangle)$

# Timing Issues

Time handling is very useful when programming business transactions, real workflow languages presently provide this feature:

1. Transactions that can be interrupted by a timeout
2. XLANG includes a notion of timed transaction as a special case of long running activity
3. BPEL allows similar behaviors by means of alarm clocks

Adding time it is possible to express more meaningful and realistic scenarios in composition

# Syntax of Timed-$\pi$

$\texttt{web}\pi$ has been equipped with an explicit mechanism for time elapsing and timeout handling. The $\texttt{web}\pi$ model of time is inspired by Berger-Honda Timed-$\pi$ skipping the idle rule plus some minor variations

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

# Semantics of Timed-$\pi$

**Definition**

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

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

$$(\text{STOP})\quad \mathtt{Timer}^{n+1}(x(\widetilde{v}).P, Q) \mid \overline{x}\,\widetilde{y} \;\rightarrow_t\; P\{\widetilde{y}/\widetilde{v}\}$$

$$(\text{IDLE})\quad P \rightarrow_t \phi_t(P) \qquad\qquad (\text{PAR})\quad \frac{P \rightarrow_t P'}{P \mid Q \rightarrow_t P' \mid \phi_t(Q)}$$

The *time-stepper function* $\phi_t$ indicates how the time passing influences the various constructs

# Sintax of $\mathtt{web}\pi$

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

# Reduction Semantics of $\mathtt{web}\pi$

## Definition

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

$$(\mathrm{COM}) \qquad \overline{x}\,\widetilde{v} \mid x(\widetilde{u}).Q \;\rightarrow\; Q\{\widetilde{v}/\widetilde{u}\}$$

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

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

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

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

# Encoding Timers

**Definition ($\pi_t$ encoding in $\mathtt{web}\pi$)**

Timers are defined by induction on $n$, for the missing cases it holds $[\![P]\!] = P$.

$$
\begin{aligned}
[\![\mathtt{Timer}^1(y(\widetilde{u}).P, Q)]\!] &= (x)(s)(\langle\!| y(\widetilde{u}).\overline{x}\,\widetilde{u} \;;\; [\![Q]\!] |\!\rangle_s^1 \mid x(\widetilde{u}).[\![P]\!]) \\
[\![\mathtt{Timer}^n(y(\widetilde{u}).P, Q)]\!] &= (x)(s)(\langle\!| y(\widetilde{u}).\overline{x}\,\widetilde{u} \;;\; [\![\mathtt{Timer}^{n-1}(y(\widetilde{u}).P, Q)]\!] |\!\rangle_s^1 \mid x(\widetilde{u}).[\![P]\!])
\end{aligned}
$$

# Barbed Similarity

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

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

**Proof.**

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

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

$\square$

# Conclusions