SCC: A Service Centered Calculus

Roberto Bruni

Dipartimento di Informatica
Università di Pisa

WS-FM 2006
Wien, Austria, September 8–9, 2006

A joint work with:

M. Boreale, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins,
U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, G. Zavattaro,

Please raise your hand and ask questions any time

SCC: A Service Centered Calculus

Roberto Bruni

Dipartimento di Informatica
Università di Pisa

WS-FM 2006
Wien, Austria, September 8–9, 2006

A joint work with:

M. Boreale, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins,
U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, G. Zavattaro,

Please raise your hand and ask questions any time

Outline

Service Oriented Computing

Features

Service-oriented computing is an emerging paradigm where services are understood as

- autonomous
- platform-independent

computational entities that can be:

- described
- published
- categorised
- discovered
- dynamically assembled for developing massively distributed, interoperable, evolvable systems.

Widespread success

Large companies invested a lot of efforts and resources to promote service delivery on a variety of computing platforms.

e-Expectations

Tomorrow, there will be a plethora of new services as required for e-government, e-business, and e-science, and other areas within the rapidly evolving Information Society.

Service Oriented Computing

Features

Service-oriented computing is an emerging paradigm where services are understood as

- autonomous
- platform-independent

computational entities that can be:

- described
- published
- categorised
- discovered
- dynamically assembled for developing massively distributed, interoperable, evolvable systems.

Widespread success

Large companies invested a lot of efforts and resources to promote service delivery on a variety of computing platforms.

e-Expectations

Tomorrow, there will be a plethora of new services as required for e-government, e-business, and e-science, and other areas within the rapidly evolving Information Society.

Service Oriented Computing

Features

Service-oriented computing is an emerging paradigm where services are understood as

- autonomous
- platform-independent

computational entities that can be:

- described
- published
- categorised
- discovered
- dynamically assembled for developing massively distributed, interoperable, evolvable systems.

Widespread success

Large companies invested a lot of efforts and resources to promote service delivery on a variety of computing platforms.

e-Expectations

Tomorrow, there will be a plethora of new services as required for e-government, e-business, and e-science, and other areas within the rapidly evolving Information Society.

SENSORIA (http://www.sensoria-ist.eu)

IST-FET Integrated Project funded by the EU in the GC Initiative (6th FP).

Industrial consortia are developing orchestration languages, targeting the standardization of Web services and XML-centric technologies, but *they lack clear semantic foundations*!

Aim

Developing a novel, comprehensive approach to the engineering of software systems for service-oriented overlay computers.

SENSORIA (http://www.sensoria-ist.eu)

IST-FET Integrated Project funded by the EU in the GC Initiative (6th FP).

Industrial consortia are developing orchestration languages, targeting the standardization of Web services and XML-centric technologies, but *they lack clear semantic foundations*!

Aim

Developing a novel, comprehensive approach to the engineering of software systems for service-oriented overlay computers.

SENSORIA (http://www.sensoria-ist.eu)

IST-FET Integrated Project funded by the EU in the GC Initiative (6th FP).

Industrial consortia are developing orchestration languages, targeting the standardization of Web services and XML-centric technologies, but *they lack clear semantic foundations*!

Aim

Developing a novel, comprehensive approach to the engineering of software systems for service-oriented overlay computers.

SENSORIA Consortium

A General Theory of Services

The strategy of SENSORIA

Integration of foundational theories, techniques, methods and tools in a pragmatic software engineering approach.

The role of process calculi

A crucial role in the project will be played by formalisms for service description that can lay the mathematical basis for analysing and experimenting with components interactions, and for combining services.

Core calculi (WP 2)

We seek for a small set of primitives that might serve as a basis for formalizing and programming service oriented applications over global computers.

A General Theory of Services

The strategy of SENSORIA

Integration of foundational theories, techniques, methods and tools in a pragmatic software engineering approach.

The role of process calculi

A crucial role in the project will be played by formalisms for service description that can lay the mathematical basis for analysing and experimenting with components interactions, and for combining services.

Core calculi (WP 2)

We seek for a small set of primitives that might serve as a basis for formalizing and programming service oriented applications over global computers.

A General Theory of Services

The strategy of SENSORIA

Integration of foundational theories, techniques, methods and tools in a pragmatic software engineering approach.

The role of process calculi

A crucial role in the project will be played by formalisms for service description that can lay the mathematical basis for analysing and experimenting with components interactions, and for combining services.

Core calculi (WP 2)

We seek for a small set of primitives that might serve as a basis for formalizing and programming service oriented applications over global computers.

Service Centered Calculus: General Principles

As an outcome of an initial study pursued during the first months of SENSORIA, we propose a process calculus that features explicit notions of

- *service definition*
- *service invocation*
- *session handling*

Sources of inspiration

We have integrated complementary aspects from

- π-calculus (naming primitives)
- Orc (pipelining and pruning of activities)
- webπ, cjoin, Sagas (primitives for LRT and compensations)

All relevant to the SOC paradigm, but so far

- not available in a single calculus
- not used in a fully disciplined way when available

Service Centered Calculus: General Principles

As an outcome of an initial study pursued during the first months of SENSORIA, we propose a process calculus that features explicit notions of

- *service definition*
- *service invocation*
- *session handling*

Sources of inspiration

We have integrated complementary aspects from

- π-calculus (naming primitives)
- Orc (pipelining and pruning of activities)
- webπ, cjoin, Sagas (primitives for LRT and compensations)

All relevant to the SOC paradigm, but so far

- not available in a single calculus
- not used in a fully disciplined way when available

Service Centered Calculus: Key Aspects

Syntax and Semantics Design

- service definition exposes: protocol + generic termination handler
- service invocation exposes: protocol + specific termination handler
- service sessions are: two-party + private
- interaction between protocols: bi-directional
- nested sessions: values can be returned outside sessions (one level up)
- local sessions termination: autonomous + on partner's request
- session termination activates partner's termination handler (if any)
- operational semantics: reduction-based

Variants

Discussed during the presentation and at the end

Service Centered Calculus: Key Aspects

Syntax and Semantics Design

- service definition exposes: protocol + generic termination handler
- service invocation exposes: protocol + specific termination handler
- service sessions are: two-party + private
- interaction between protocols: bi-directional
- nested sessions: values can be returned outside sessions (one level up)
- local sessions termination: autonomous + on partner's request
- session termination activates partner's termination handler (if any)
- operational semantics: reduction-based

Variants

Discussed during the presentation and at the end

In This Talk

Advice

The formal presentation of SCC involves some key notational and technical solutions.

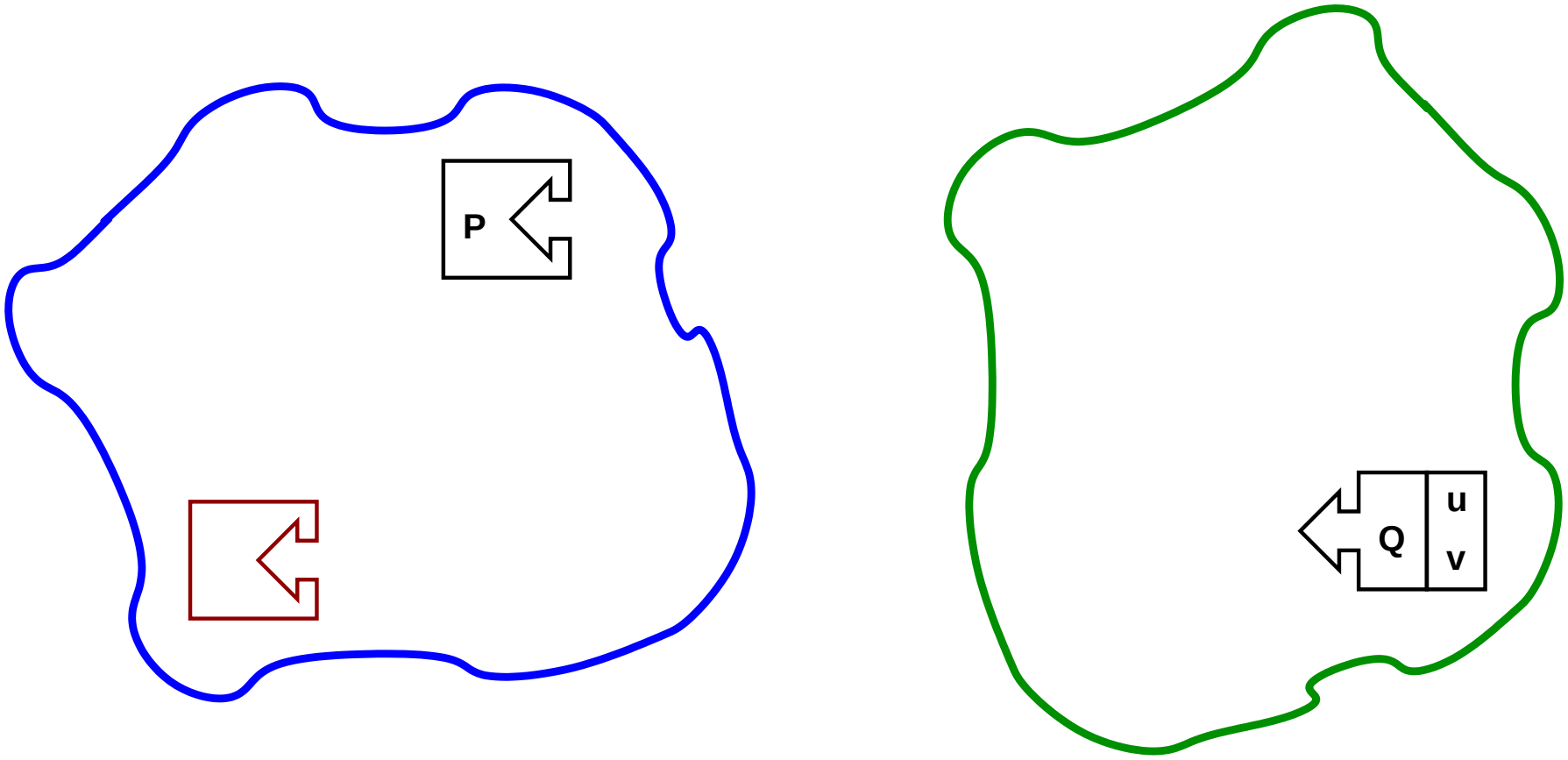
Roadmap

We will give a gentle, step-by-step presentation of the various ingredients:

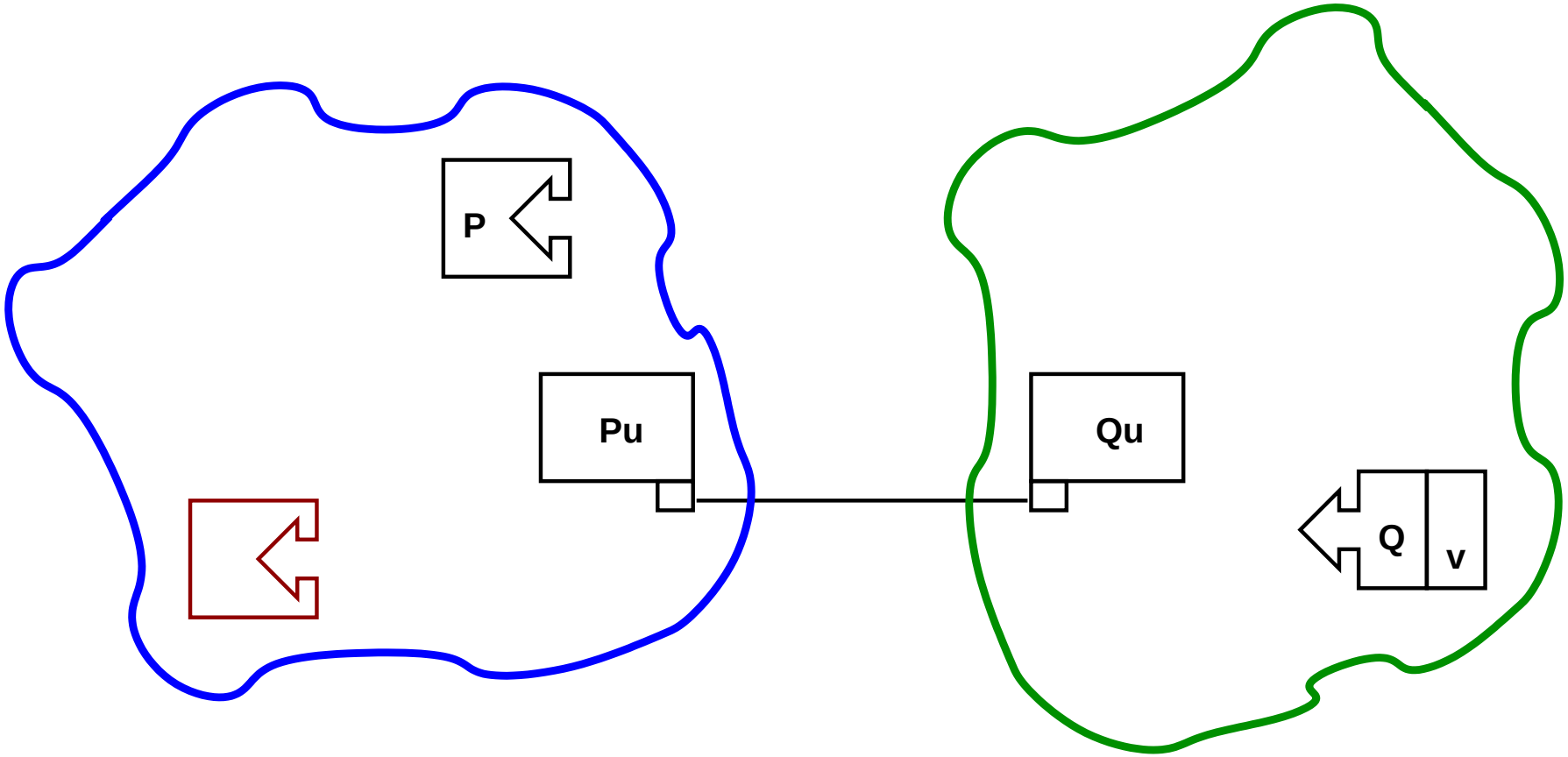
1. first a reduced fragment, called Persistent Session Calculus (PSC), then full Service Centered calculus (SCC)
2. a number of programming samples that demonstrate flexibility of the chosen set of primitives (we follow the "everything is a service" paradigm)
3. we show the implementability of Orc primitives (not available in π-calculus) and, as a consequence, of van der Aalst's most common Workflow Patterns

Outline

Service Invocation, Graphically



Bidirectional Session, Graphically

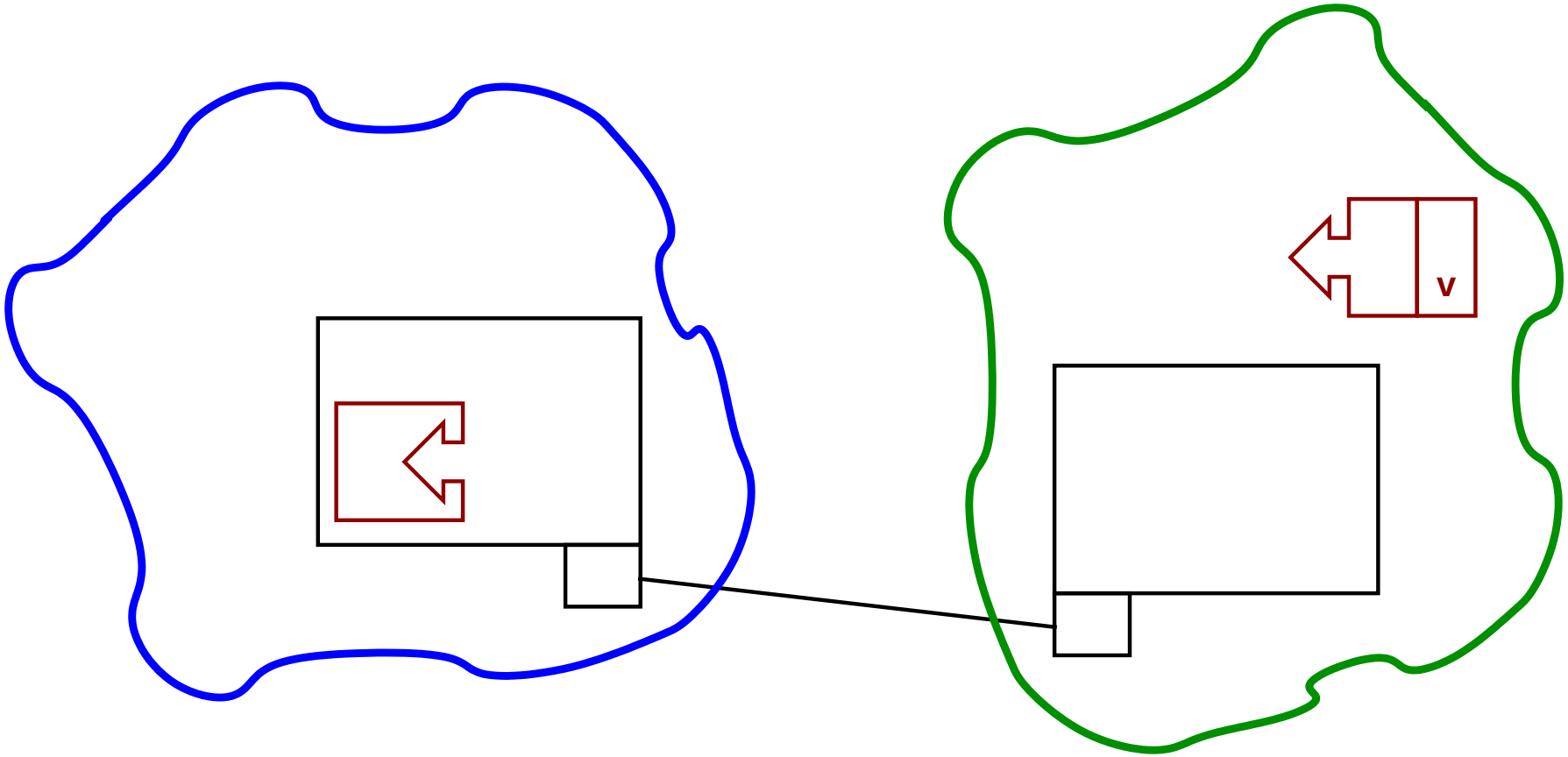


Intra-Session Communication, Graphically

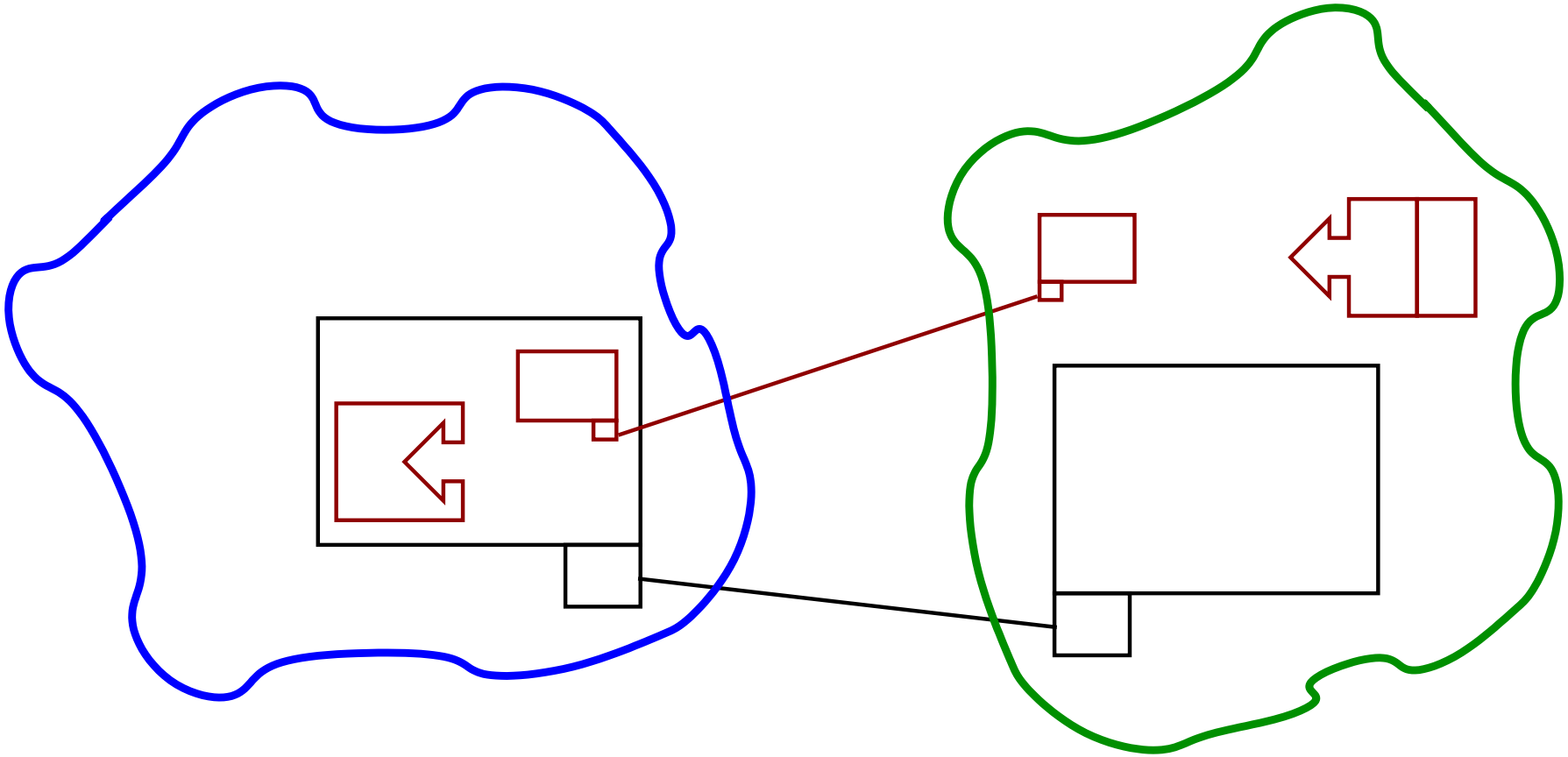
Intra-Session Communication, Graphically

Intra-Session Communication, Graphically

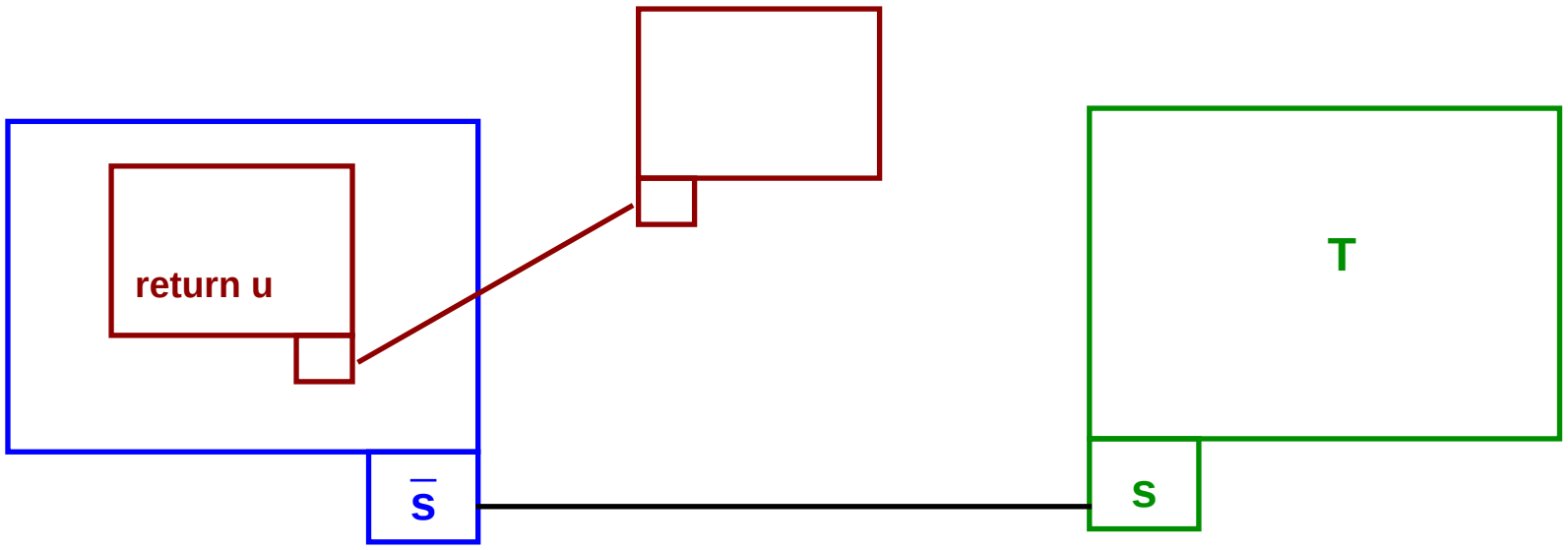
Nested Services and Multi-Sessions, Graphically



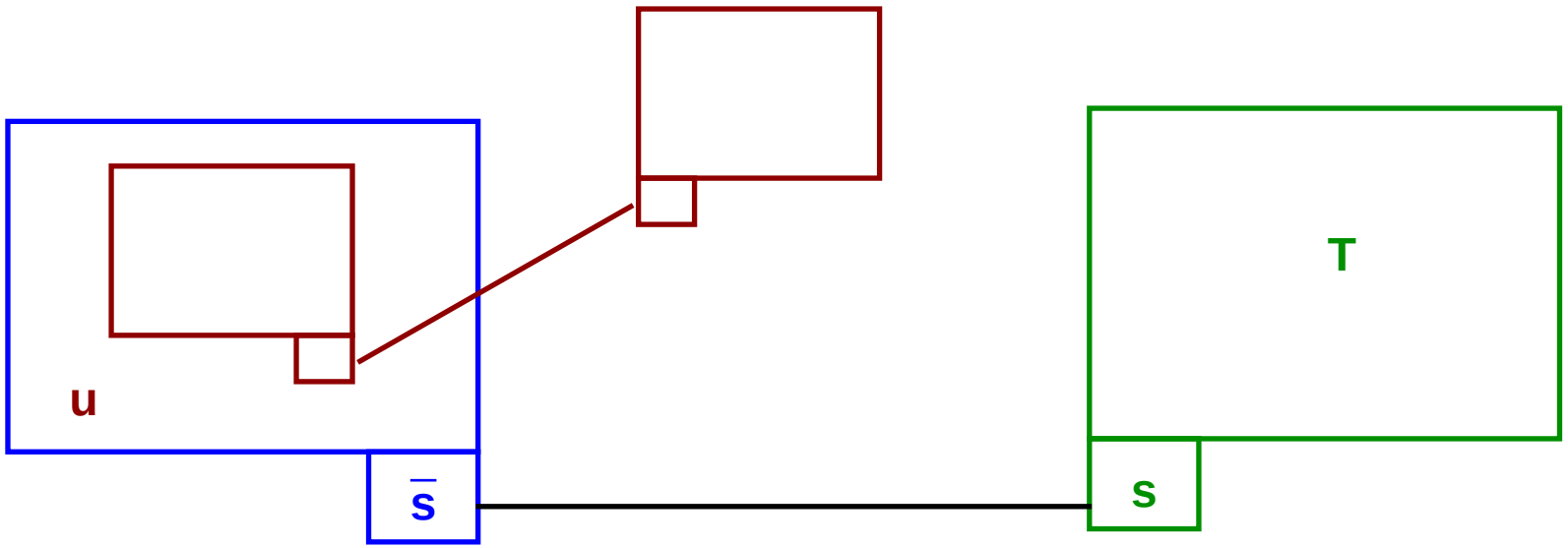
Nested Services and Multi-Sessions, Graphically



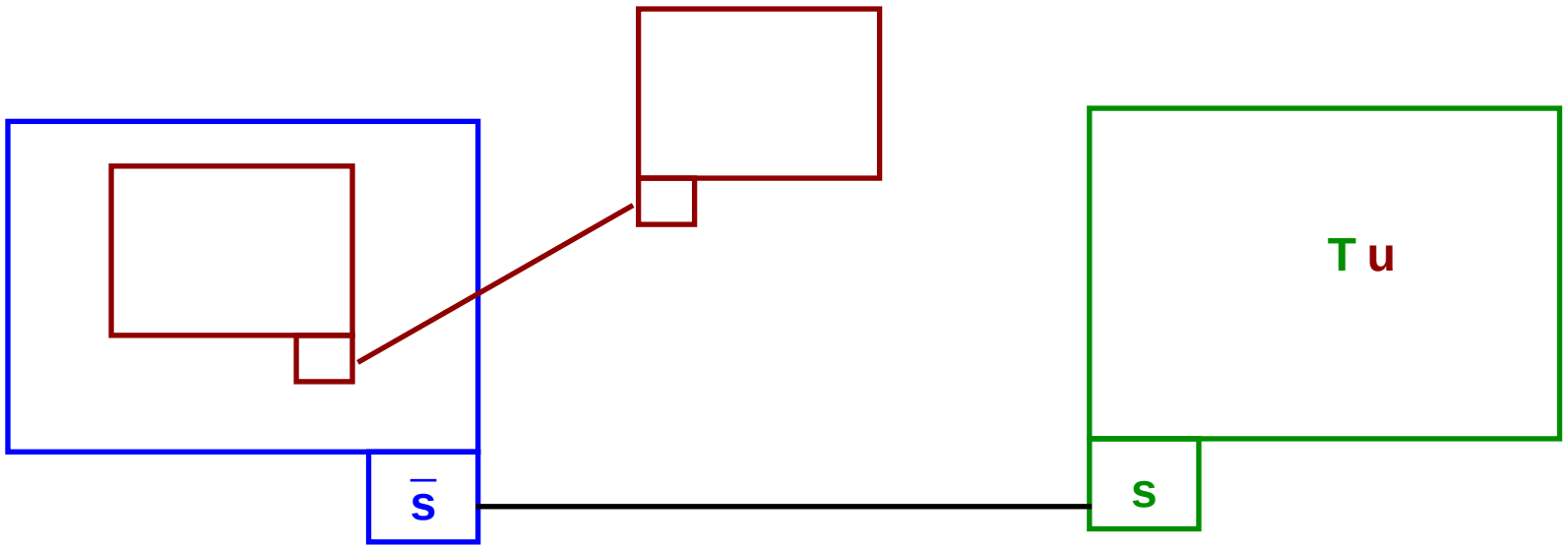
Returning Values, Graphically



Returning Values, Graphically



Returning Values, Graphically



Outline

Service Definition

Service definition

$$s \Rightarrow (x)P$$

- s is the service name
- x is the formal parameter
- P is the actual implementation of the service.

Examples: Successor and prime teller

$$\mathtt{succ} \Rightarrow (x)x + 1$$

Received an integer communicates back its successor.

$$\mathtt{prime} \Rightarrow (n)P$$

Received an integer n communicates back the n-th prime number.

Service Definition

Service definition

$$s \Rightarrow (x)P$$

- s is the service name
- x is the formal parameter
- P is the actual implementation of the service.

Examples: Successor and prime teller

$$\texttt{succ} \Rightarrow (x)x + 1$$

Received an integer communicates back its successor.

$$\texttt{prime} \Rightarrow (n)P$$

Received an integer n communicates back the n-th prime number.

Service Invocation

Service invocation

$$s\{(x)P\} \Leftarrow Q$$

- each new value v produced by the client Q will trigger a new invocation of service s (like Orc sequencing $Q > x > P$)
- for each invocation, a suitable instance $P\{^{v}/_{x}\}$ of the process P, implements the client-side protocol

Example: A sample client

$$\texttt{prime}\{(x)(y)\text{return } y\} \Leftarrow 5$$

Shorthand notation

The client side makes no use of the formal parameter x: we abbreviate it as

$$\texttt{prime}\{(-)(y)\text{return } y\} \Leftarrow 5$$

Service Invocation

Service invocation

$$s\{(x)P\} \Leftarrow Q$$

- each new value v produced by the client Q will trigger a new invocation of service s (like Orc sequencing $Q > x > P$)
- for each invocation, a suitable instance $P\{^{v}/_{x}\}$ of the process P, implements the client-side protocol

Example: A sample client

$$\texttt{prime}\{(x)(y)\text{return } y\} \Leftarrow 5$$

Shorthand notation

The client side makes no use of the formal parameter x: we abbreviate it as

$$\texttt{prime}\{(-)(y)\text{return } y\} \Leftarrow 5$$

Service Activation

Service activation

$$\mathbb{C}[\![\, s \Rightarrow (x)P \,]\!] \mid \mathbb{D}[\![\, s\{(y)P'\} \Leftarrow (Q \mid u.R) \,]\!] \;\rightarrow\; (\nu r)\left(\mathbb{C}[\![\, r \triangleright P\{^{u}/_{x}\} \mid s \Rightarrow (x)P \,]\!] \mid \mathbb{D}[\![\, \overline{r} \triangleright P'\{^{u}/_{y}\} \mid s\{(y)P'\} \Leftarrow (Q \mid R) \,]\!] \right)$$

if r is fresh and u, s not bound by $\mathbb{C}$,$\mathbb{D}$

A service invocation causes activation of a new session:

- dual fresh identifiers, r and $\overline{r}$, name the two sides of the session
- client and service protocols run each at the proper side of the session

Example: Asking for prime numbers

The invocation of service `prime` triggers the session

$$(\nu r)\big(\ldots \overline{r} \triangleright P\{^{5}/_{n}\} \ldots \mid \ldots r \triangleright (z)\mathsf{return}\, z \ldots \big)$$

The client waits for a value from the server (11) to be substituted for z

Service Activation

Service activation

$$\begin{array}{l}\mathbb{C}[\![\, s \Rightarrow (x)P \,]\!] \mid \\ \mathbb{D}[\![\, s\{(y)P'\} \Leftarrow (Q \mid u.R) \,]\!]\end{array} \quad \rightarrow \quad (\nu r)\left(\begin{array}{l}\mathbb{C}[\![\, r \triangleright P\{^{u}/_{x}\} \mid s \Rightarrow (x)P \,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \triangleright P'\{^{u}/_{y}\} \mid s\{(y)P'\} \Leftarrow (Q \mid R) \,]\!]\end{array}\right)$$

if r is fresh and u, s not bound by $\mathbb{C}$,$\mathbb{D}$

A service invocation causes activation of a new session:

- dual fresh identifiers, r and $\overline{r}$, name the two sides of the session
- client and service protocols run each at the proper side of the session

Example: Asking for prime numbers

The invocation of service `prime` triggers the session

$$(\nu r)\big(\ldots \overline{r} \triangleright P\{^{5}/_{n}\} \ldots \mid \ldots r \triangleright (z)\mathsf{return}\, z \ldots\big)$$

The client waits for a value from the server (11) to be substituted for z

Service Activation

Service activation

$$\begin{array}{l}\mathbb{C}[\![\, s \Rightarrow (x)P \,]\!] \mid \\ \mathbb{D}[\![\, s\{(y)P'\} \Leftarrow (Q|u.R) \,]\!]\end{array} \rightarrow (\nu r)\left(\begin{array}{l}\mathbb{C}[\![\, r \triangleright P\{^{u}/_{x}\} \mid s \Rightarrow (x)P \,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \triangleright P'\{^{u}/_{y}\} \mid s\{(y)P'\} \Leftarrow (Q|R) \,]\!]\end{array}\right)$$

if r is fresh and u, s not bound by $\mathbb{C}$,$\mathbb{D}$

A service invocation causes activation of a new session:

- dual fresh identifiers, r and $\overline{r}$, name the two sides of the session
- client and service protocols run each at the proper side of the session

Example: Asking for prime numbers

The invocation of service `prime` triggers the session

$$(\nu r)\big(\ldots \overline{r} \triangleright P\{^{5}/_{n}\} \ldots \mid \ldots r \triangleright (z)\mathsf{return}\, z \ldots\big)$$

The client waits for a value from the server (11) to be substituted for z

Service Activation

Service activation

$$\begin{array}{l}\mathbb{C}[\![\, s \Rightarrow (x)P \,]\!] \mid \\ \mathbb{D}[\![\, s\{(y)P'\} \Leftarrow (Q|u.R) \,]\!]\end{array} \rightarrow (\nu r)\left(\begin{array}{l}\mathbb{C}[\![\, r \rhd P\{^{u}/_{x}\} \mid s \Rightarrow (x)P \,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \rhd P'\{^{u}/_{y}\} \mid s\{(y)P'\} \Leftarrow (Q|R) \,]\!]\end{array}\right)$$

if r is fresh and u, s not bound by $\mathbb{C}$,$\mathbb{D}$

A service invocation causes activation of a new session:

- dual fresh identifiers, r and $\overline{r}$, name the two sides of the session
- client and service protocols run each at the proper side of the session

Example: Asking for prime numbers

The invocation of service `prime` triggers the session

$$(\nu r)\big(\,\ldots \overline{r} \rhd P\{^{5}/_{n}\} \ldots \mid \ldots r \rhd (z)\mathsf{return}\, z \ldots\,\big)$$

The client waits for a value from the server (11) to be substituted for z

Session Communication

Session communication

$$\begin{array}{l}\mathbb{C}[\![\, r \rhd (\, P \mid u.Q \,)\,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \rhd (\, R \mid (z)S \,)\,]\!]\end{array} \quad \longrightarrow \quad \begin{array}{l}\mathbb{C}[\![\, r \rhd (\, P \mid Q \,)\,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \rhd (\, R \mid S\{^{u}/_{z}\} \,)\,]\!]\end{array}$$

if u, r not bound by $\mathbb{C}$,$\mathbb{D}$

Within sessions, communication is bi-directional, in the sense that the interacting protocols can exchange data in both directions.

Example: 5th prime evaluated and communicated

After the value 11 has been computed, it can be communicated:

$$(\nu r)(\, \ldots \overline{r} \rhd 11 \ldots \mid \ldots r \rhd (z)\mathsf{return}\, z \ldots\,)$$

Session Communication

Session communication

$$\begin{array}{l}\mathbb{C}[\![\, r \rhd (\, P \mid u.Q \,)\,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \rhd (\, R \mid (z)S \,)\,]\!]\end{array} \quad \longrightarrow \quad \begin{array}{l}\mathbb{C}[\![\, r \rhd (\, P \mid Q \,)\,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \rhd (\, R \mid S\{^{u}/_{z}\} \,)\,]\!]\end{array}$$

if u, r not bound by $\mathbb{C}$,$\mathbb{D}$

Within sessions, communication is bi-directional, in the sense that the interacting protocols can exchange data in both directions.

Example: 5th prime evaluated and communicated

After the value 11 has been computed, it can be communicated:

$$(\nu r)(\, \ldots \overline{r} \rhd 11 \ldots \mid \ldots r \rhd (z)\mathsf{return}\, z \ldots\,)$$

$$(\nu r)(\, \ldots \overline{r} \rhd 0 \ldots \mid \ldots r \rhd \mathsf{return}\, 11 \ldots\,)$$

Session Communication

Session communication

$$\begin{array}{l}\mathbb{C}[\![\, r \triangleright (\, P \mid u.Q \,)\,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \triangleright (\, R \mid (z)S \,)\,]\!]\end{array} \quad \rightarrow \quad \begin{array}{l}\mathbb{C}[\![\, r \triangleright (\, P \mid Q \,)\,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \triangleright (\, R \mid S\{^{u}/_{z}\} \,)\,]\!]\end{array}$$

if u, r not bound by $\mathbb{C}$,$\mathbb{D}$

Within sessions, communication is bi-directional, in the sense that the interacting protocols can exchange data in both directions.

Example: 5th prime evaluated and communicated

After the value 11 has been computed, it can be communicated:

$$(\nu r)(\, \ldots \overline{r} \triangleright 11 \ldots \mid \ldots r \triangleright (z)\mathsf{return}\, z \ldots\,)$$

$$(\nu r)(\, \ldots \overline{r} \triangleright \mathbf{0} \ldots \mid \ldots r \triangleright \mathsf{return}\, 11 \ldots\,)$$

Session Communication

Session communication

$$\begin{array}{l}\mathbb{C}[\![\, r \triangleright (\, P \mid u.Q \,) \,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \triangleright (\, R \mid (z)S \,) \,]\!]\end{array} \quad \longrightarrow \quad \begin{array}{l}\mathbb{C}[\![\, r \triangleright (\, P \mid Q \,) \,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \triangleright (\, R \mid S\{^{u}/_{z}\} \,) \,]\!]\end{array}$$

if u, r not bound by $\mathbb{C}$,$\mathbb{D}$

Within sessions, communication is bi-directional, in the sense that the interacting protocols can exchange data in both directions.

Example: 5th prime evaluated and communicated

After the value 11 has been computed, it can be communicated:

$$(\nu r)\big(\, \ldots \overline{r} \triangleright 11 \ldots \mid \ldots r \triangleright (z)\mathsf{return}\, z \ldots \big)$$

$$(\nu r)\big(\, \ldots \overline{r} \triangleright \mathbf{0} \ldots \mid \ldots r \triangleright \mathsf{return}\, 11 \ldots \big)$$

Session Communication

Session communication

$$\begin{array}{l}\mathbb{C}[\![\, r \triangleright (\, P \mid u.Q \,) \,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \triangleright (\, R \mid (z)S \,) \,]\!]\end{array} \quad \longrightarrow \quad \begin{array}{l}\mathbb{C}[\![\, r \triangleright (\, P \mid Q \,) \,]\!] \mid \\ \mathbb{D}[\![\, \overline{r} \triangleright (\, R \mid S\{^{u}/_{z}\} \,) \,]\!]\end{array}$$

if u, r not bound by $\mathbb{C}$,$\mathbb{D}$

Within sessions, communication is bi-directional, in the sense that the interacting protocols can exchange data in both directions.

Example: 5th prime evaluated and communicated

After the value 11 has been computed, it can be communicated:

$$(\nu r)\big(\, \ldots \overline{r} \triangleright 11 \ldots \mid \ldots r \triangleright (z)\mathsf{return}\, z \ldots \big)$$

$$(\nu r)\big(\, \ldots \overline{r} \triangleright \mathbf{0} \ldots \mid \ldots r \triangleright \mathsf{return}\, 11 \ldots \big)$$

Session Returning Values

Session returning values

$$r \triangleright (\, P \,|\, \text{return } u.Q \,) \;\longrightarrow\; u \;|\; r \triangleright (\, P \,|\, Q \,)$$

Values can be returned outside the session to the enclosing environment and used for invoking other services.

Example: Returning the 5th prime number

$$(\nu r)\big(\, \ldots \overline{r} \triangleright \mathbf{0} \ldots \,|\, \ldots 11 \,|\, r \triangleright \mathbf{0} \ldots \,\big)$$

A taste of structural congruence

Terminated protocols are immaterial

$$r \triangleright \mathbf{0} \quad \equiv \quad \mathbf{0}$$

Session Returning Values

Session returning values

$$r \triangleright (\, P \mid \text{return } u.Q \,) \;\longrightarrow\; u \mid r \triangleright (\, P \mid Q \,)$$

Values can be returned outside the session to the enclosing environment and used for invoking other services.

Example: Returning the 5th prime number

$$(\nu r)\big(\, \ldots \overline{r} \triangleright \mathbf{0} \ldots \mid \ldots 11 \mid r \triangleright \mathbf{0} \ldots \,\big)$$

A taste of structural congruence

Terminated protocols are immaterial

$$r \triangleright \mathbf{0} \quad \equiv \quad \mathbf{0}$$

Session Returning Values

Session returning values

$$r \rhd (\, P \,|\, \text{return } u.Q \,) \;\longrightarrow\; u \;|\; r \rhd (\, P \,|\, Q \,)$$

Values can be returned outside the session to the enclosing environment and used for invoking other services.

Example: Returning the 5th prime number

$$(\nu r)\big(\, \ldots \overline{r} \rhd \mathbf{0} \ldots \,|\, \ldots 11 \,|\, r \rhd \mathbf{0} \ldots \,\big)$$

A taste of structural congruence

Terminated protocols are immaterial

$$r \rhd \mathbf{0} \quad \equiv \quad \mathbf{0}$$

Functional Flavour

A "functional" protocol

A common pattern of service invocation is:

$$s\{(-)(y)\text{return}\, y\} \Leftarrow P$$

where s is invoked on every value that P produces

Shorthand notation

$$s \Leftarrow P$$

Example: Successor of a prime

We write

$$\texttt{succ} \Leftarrow (\texttt{prime} \Leftarrow 5)$$

instead of $\texttt{succ}\{(-)(w)\text{return}\, w\} \Leftarrow (\texttt{prime}\{(-)(y)\text{return}\, y\} \Leftarrow 5)$

Functional Flavour

A "functional" protocol

A common pattern of service invocation is:

$$s\{(-)(y)\text{return}\, y\} \Leftarrow P$$

where s is invoked on every value that P produces

Shorthand notation

$$s \Leftarrow P$$

Example: Successor of a prime

We write

$$\texttt{succ} \Leftarrow (\texttt{prime} \Leftarrow 5)$$

instead of $\texttt{succ}\{(-)(w)\text{return}\, w\} \Leftarrow (\texttt{prime}\{(-)(y)\text{return}\, y\} \Leftarrow 5)$

Functional Flavour

A "functional" protocol

A common pattern of service invocation is:

$$s\{(-)(y)\text{return } y\} \Leftarrow P$$

where s is invoked on every value that P produces

Shorthand notation

$$s \Leftarrow P$$

Example: Successor of a prime

We write

$$\texttt{succ} \Leftarrow (\texttt{prime} \Leftarrow 5)$$

instead of $\texttt{succ}\{(-)(w)\text{return } w\} \Leftarrow (\texttt{prime}\{(-)(y)\text{return } y\} \Leftarrow 5)$

Blind Invocation

Vacuous protocol

If no reply is expected from a service, the client can employ a vacuous protocol

$$a\{(-)\mathbf{0}\} \Leftarrow P$$

Shorthand notation

$$a\{\} \Leftarrow P$$

Example: Printing values

A client invokes the service `prime` and then prints the result:

$$\texttt{print}\{\} \Leftarrow (\,\texttt{prime} \Leftarrow 5\,)$$

In this case, the service `print` is invoked with vacuous protocol $(z)\mathbf{0}$

Blind Invocation

Vacuous protocol

If no reply is expected from a service, the client can employ a vacuous protocol

$$a\{(-)\mathbf{0}\} \Leftarrow P$$

Shorthand notation

$$a\{\} \Leftarrow P$$

Example: Printing values

A client invokes the service `prime` and then prints the result:

$$\texttt{print}\{\} \Leftarrow (\,\texttt{prime} \Leftarrow 5\,)$$

In this case, the service `print` is invoked with vacuous protocol $(z)\mathbf{0}$

Blind Invocation

Vacuous protocol

If no reply is expected from a service, the client can employ a vacuous protocol

$$a\{(-)\mathbf{0}\} \Leftarrow P$$

Shorthand notation

$$a\{\} \Leftarrow P$$

Example: Printing values

A client invokes the service `prime` and then prints the result:

$$\texttt{print}\{\} \Leftarrow (\,\texttt{prime} \Leftarrow 5\,)$$

In this case, the service `print` is invoked with vacuous protocol $(z)\mathbf{0}$

A Taste of SCC: Session Termination

Handling interruption

A protocol (on both sides of a session) can be interrupted (e.g. due to the occurrence of an unexpected event), and interruption can be notified to a suitable handler at the partner site.

Example: Printing values with faulty printers

Below, a suitable service `fault` handles printer failures:

$$\texttt{print}\{\} \Leftarrow_{\texttt{fault}} (\,\texttt{prime} \Leftarrow 5\,)$$

PSC Syntax

Grammar

We presuppose a countable set $\mathcal{N}$ of names $a, b, c, \ldots, r, s, \ldots, x, y, \ldots$, with a bijection $\bar{\cdot}$ on $\mathcal{N}$ s.t. $\overline{\bar{a}} = a$ for each name a.

P, Q	$::=$	$\mathbf{0}$	Nil
		$\mid a.P$	Concretion (pass a to session partner)
		$\mid (x)P$	Abstraction (take from session partner)
		$\mid \text{return}\, a.P$	Return Value (out of current session)
		$\mid s \Rightarrow (x)P$	Service Definition
		$\mid s\{(x)P\} \Leftarrow Q$	Service Invocation
		$\mid r \rhd P$	Session Side
		$\mid P \mid Q$	Parallel Composition
		$\mid (\nu a)P$	New Name

(operators are listed in decreasing order of precedence)

PSC Structural Congruence

Axioms

$$
\begin{array}{rcll}
P & \equiv & Q & \text{if } P =_\alpha Q \\
(P \mid Q) \mid R & \equiv & P \mid (Q \mid R) & \\
P \mid Q & \equiv & Q \mid P & \\
P \mid \mathbf{0} & \equiv & P & \\
(\nu x)(\nu y)P & \equiv & (\nu y)(\nu x)P & \\
(\nu x)\mathbf{0} & \equiv & \mathbf{0} & \\
P \mid (\nu x)Q & \equiv & (\nu x)(P \mid Q) & \text{if } x \notin fn(P) \\
r \rhd (\nu x)P & \equiv & (\nu x)(r \rhd P) & \text{if } x \notin \{r, \overline{r}\} \\
s\{(x)P\} \Leftarrow (\nu y)Q & \equiv & (\nu y)(r\{(x)P\} \Leftarrow Q) & \text{if } y \notin fn((x)P) \cup \{r, \overline{r}\} \\
r \rhd \mathbf{0} & \equiv & \mathbf{0} &
\end{array}
$$

PSC Operational Semantics

Active contexts

$$\mathbb{C}, \mathbb{D} \;::=\; [\![\cdot]\!] \mid \mathbb{C} \,|\, P \mid a\{(x)P\} \Leftarrow \mathbb{C} \mid a \rhd \mathbb{C} \mid (\nu a)\mathbb{C}$$

An active context is a process with a hole $[\![\cdot]\!]$ in an active position.
We denote by $\mathbb{C}[\![P]\!]$ the process obtained by filling the hole in $\mathbb{C}$ with P

Reductions

$$\begin{array}{l} \mathbb{C}[\![\, s \Rightarrow (x)P \,]\!] \,| \\ \mathbb{D}[\![\, s\{(y)P'\} \Leftarrow (Q|u.R) \,]\!] \end{array} \;\rightarrow\; (\nu r) \left(\begin{array}{l} \mathbb{C}[\![\, r \rhd P\{^u/_x\} \mid s \Rightarrow (x)P \,]\!] \,| \\ \mathbb{D}[\![\, \overline{r} \rhd P'\{^u/_y\} \mid s\{(y)P'\} \Leftarrow (Q|R) \,]\!] \end{array} \right)$$

if r is fresh and u, s not bound by $\mathbb{C}$,$\mathbb{D}$

$$\mathbb{C}[\![\, r \rhd (P|u.Q) \,]\!] \mid \mathbb{D}[\![\, \overline{r} \rhd (R|(z)S) \,]\!] \;\rightarrow\; \mathbb{C}[\![\, r \rhd (P|Q) \,]\!] \mid \mathbb{D}[\![\, \overline{r} \rhd (R|S\{^u/_z\}) \,]\!]$$

if u, r not bound by $\mathbb{C}$,$\mathbb{D}$

$$r \rhd (P|\mathsf{return}\, u.Q) \;\rightarrow\; u \,|\, r \rhd (P|Q)$$

$$\mathbb{C}[\![P]\!] \;\rightarrow\; \mathbb{C}[\![P']\!] \quad \text{if } P \equiv Q,\; Q \rightarrow Q',\; Q' \equiv P'$$

PSC: A Brief Discussion

Persistency

We call it PSC for *persistent session calculus*:

- sessions can be established
- a session can be garbage collected when the protocol has run entirely,
- but sessions can neither be aborted nor closed by one of the parties

A note on well-formedness

A process is *well-formed* if (assuming by α-conversion that all its bound names are different from each other and from the free names):

- each session name r occurs only once ($r \triangleright \mathbf{0}$ is immaterial)
- it is allowed to have both sessions $r \triangleright Q$ and $\overline{r} \triangleright Q'$.

The use of dual names is not stricly necessary, but we prefer to keep this distinction to make evident that once the protocol is started there might still be some reasons for distinguishing the two side ends (e.g., types).

PSC Examples: Recursion - I

Shorthand notation

We presuppose a distinct name $\bullet$ to be used as a unit value.

Clock (service side)

Service invocations can be nested recursively inside a service definition:

$$\texttt{clock} \Rightarrow (-) \left(\begin{array}{l} \texttt{return tick} \\ \mid \quad \texttt{clock}\{\} \Leftarrow \bullet \end{array} \right)$$

Invoked with $\texttt{clock}\{\} \Leftarrow \bullet$, produces an infinite number of `ticks`...
but just on the service-side!

PSC Examples: Recursion - I

Shorthand notation

We presuppose a distinct name $\bullet$ to be used as a unit value.

Clock (service side)

Service invocations can be nested recursively inside a service definition:

$$\texttt{clock} \Rightarrow (-) \left(\begin{array}{l} \texttt{return tick} \\ \mid \quad \texttt{clock}\{\} \Leftarrow \bullet \end{array} \right)$$

Invoked with $\texttt{clock}\{\} \Leftarrow \bullet$, produces an infinite number of `ticks`...
but just on the service-side!

PSC Examples: Recursion - II

Clock (client side)

To produce the `ticks` on a specific location different from the service-side, the service to be invoked can be written as

$$remoteClock \Rightarrow (s) \left(\begin{array}{ll} & s\{\} \Leftarrow \texttt{tick} \\ | & remoteClock\{\} \Leftarrow s \end{array} \right)$$

and a local publishing service

$$pub \Rightarrow (t)\text{return } t$$

must be located where the `ticks` must be produced.
Then invoke the service as below:

$$remoteClock\{\} \Leftarrow pub$$

PSC Examples: Stream Connection - 1

Observation

If P is a process that produces a stream of values then the composition $q \Leftarrow P$ invokes q infinitely often.

Question

The service seen at the end of the previous example produces an unbounded stream of values.
Is it possible to deploy some sort of pipeline between two services p and q in such a way that q is invoked for each value produced by p?
Or equivalently, is it possible to design a client-side protocol for collecting all the values returned by p?

PSC Examples: Stream Connection - II

Trivial recursion does not work!

One might think to exploit recursion to deploy local receivers of the form $(x)\text{return}\, x$, but the implicit nesting of sessions would cause all such receivers to collect values only from different sessions than the original one.

No Replicator!

Extending the syntax with π-calculus like replicator $!P$:

$$pipe = (-)!(x)\text{return}\, x$$

No Code Passing!

Extending the syntax with $\text{return}\, P.Q$, whose semantics is:

$$r \rhd (R|\text{return}\, P.Q) \;\rightarrow\; P \,|\, r \rhd\; (R|Q)$$

Replication can then be coded as follows:

$$!P = (\nu\, rec)\big(\; rec \Rightarrow (-)(\,\text{return}\, P \,|\, rec\{\} \Leftarrow \bullet\,) \,|\, rec\{\} \Leftarrow \bullet\;\big)$$

PSC Examples: Stream Connection - II

Trivial recursion does not work!

One might think to exploit recursion to deploy local receivers of the form $(x)\text{return}\, x$, but the implicit nesting of sessions would cause all such receivers to collect values only from different sessions than the original one.

No Replicator!

Extending the syntax with π-calculus like replicator $!P$:

$$pipe = (-)!(x)\text{return}\, x$$

No Code Passing!

Extending the syntax with $\text{return}\, P.Q$, whose semantics is:

$$r \rhd (R|\text{return}\, P.Q) \;\rightarrow\; P \,|\, r \rhd\; (R|Q)$$

Replication can then be coded as follows:

$$!P = (\nu\, rec)\big(\; rec \Rightarrow (-)(\,\text{return}\, P \,|\, rec\{\} \Leftarrow \bullet\,) \,|\, rec\{\} \Leftarrow \bullet\;\big)$$

PSC Examples: Stream Connection - II

Trivial recursion does not work!

One might think to exploit recursion to deploy local receivers of the form $(x)\text{return}\, x$, but the implicit nesting of sessions would cause all such receivers to collect values only from different sessions than the original one.

No Replicator!

Extending the syntax with π-calculus like replicator $!P$:

$$pipe = (-)!(x)\text{return}\, x$$

No Code Passing!

Extending the syntax with $\text{return}\, P.Q$, whose semantics is:

$$r \rhd (R|\text{return}\, P.Q) \;\rightarrow\; P \,|\, r \rhd \; (R|Q)$$

Replication can then be coded as follows:

$$!P = (\nu\, rec)\big(\; rec \Rightarrow (-)(\,\text{return}\, P \,|\, rec\{\} \Leftarrow \bullet\,) \,|\, rec\{\} \Leftarrow \bullet\;\big)$$

PSC Examples: Stream Connection - III

We can use a publisher!

Without extending the syntax of the calculus, a solution is to install a local publishing service like *pub* above, which must be passed to *p* (and properly used therein).

Conference announcements

For instance, if *EATCS* and *EAPLS* return streams of conference announcements on the received service name, then

$$emailMe\{\} \Leftarrow \left(\begin{array}{l} pub \Rightarrow (s)\text{return } s \\ \mid \quad EATCS\{\} \Leftarrow pub \\ \mid \quad EAPLS\{\} \Leftarrow pub \end{array} \right)$$

will send you all the announcements collected from *EATCS* and *EAPLS*. More concisely, this can be equivalently written as

$$EATCS\{\} \Leftarrow emailMe \mid EAPLS\{\} \Leftarrow emailMe.$$

PSC Examples: Stream Connection - III

We can use a publisher!

Without extending the syntax of the calculus, a solution is to install a local publishing service like *pub* above, which must be passed to p (and properly used therein).

Conference announcements

For instance, if *EATCS* and *EAPLS* return streams of conference announcements on the received service name, then

$$emailMe\{\} \Leftarrow \left(\begin{array}{ll} & pub \Rightarrow (s)\text{return } s \\ | & EATCS\{\} \Leftarrow pub \\ | & EAPLS\{\} \Leftarrow pub \end{array} \right)$$

will send you all the announcements collected from *EATCS* and *EAPLS*.

More concisely, this can be equivalently written as

$$EATCS\{\} \Leftarrow emailMe \mid EAPLS\{\} \Leftarrow emailMe.$$

PSC Examples: Stream Connection - III

We can use a publisher!

Without extending the syntax of the calculus, a solution is to install a local publishing service like *pub* above, which must be passed to p (and properly used therein).

Conference announcements

For instance, if *EATCS* and *EAPLS* return streams of conference announcements on the received service name, then

$$emailMe\{\} \Leftarrow \left(\begin{array}{ll} & pub \Rightarrow (s)\mathsf{return}\, s \\ | & EATCS\{\} \Leftarrow pub \\ | & EAPLS\{\} \Leftarrow pub \end{array} \right)$$

will send you all the announcements collected from *EATCS* and *EAPLS*. More concisely, this can be equivalently written as

$$EATCS\{\} \Leftarrow emailMe \mid EAPLS\{\} \Leftarrow emailMe.$$

More on PSC

If you want to see more on PSC...

The paper in the proceedings includes:

- a `bookRoom` service, that exploits a more elaborated (two-way) client-side protocol
- the encoding of lazy λ-calculus in PSC
- the encoding of PSC in π-calculus
  - the vice versa is not easy because of sessioning
  - note also that service definition and invocation are not prefixes

Outline

From PSC to SCC - I

Once the two protocols $r \triangleright P_1$ at client-side and $\overline{r} \triangleright P_2$ at service-side are activated, the session is garbage collected by the structural congruence only when the protocols reduce to $\mathbf{0}$.

Many sessions can never reduce to $\mathbf{0}$, e.g., those containing service definitions!

Also, one may want to explicit program session termination, for instance in order to implement *cancellation workflow patterns* or Orc's *asymmetric parallel* or to manage *abnormal events*.

Termination handler

The termination handler service is associated to sessions on their instantiation. The intuition that we follow is that the termination of the session on one side, should be communicated to the opposite side.

From PSC to SCC - I

Once the two protocols $r \triangleright P_1$ at client-side and $\overline{r} \triangleright P_2$ at service-side are activated, the session is garbage collected by the structural congruence only when the protocols reduce to $\mathbf{0}$.

Many sessions can never reduce to $\mathbf{0}$, e.g., those containing service definitions!

Also, one may want to explicit program session termination, for instance in order to implement *cancellation workflow patterns* or Orc's *asymmetric parallel* or to manage *abnormal events*.

Termination handler

The termination handler service is associated to sessions on their instantiation. The intuition that we follow is that the termination of the session on one side, should be communicated to the opposite side.

From PSC to SCC - I

Once the two protocols $r \triangleright P_1$ at client-side and $\overline{r} \triangleright P_2$ at service-side are activated, the session is garbage collected by the structural congruence only when the protocols reduce to $\mathbf{0}$.

Many sessions can never reduce to $\mathbf{0}$, e.g., those containing service definitions!

Also, one may want to explicit program session termination, for instance in order to implement *cancellation workflow patterns* or Orc's *asymmetric parallel* or to manage *abnormal events*.

Termination handler

The termination handler service is associated to sessions on their instantiation. The intuition that we follow is that the termination of the session on one side, should be communicated to the opposite side.

From PSC to SCC - II

Extending sessions

- A service name k, identifying the so-called *termination handler* service, can be associated to each session: $r \rhd_k P$
- The first time the protocol P running inside the session invokes such a service k, the session is closed

Extending services: A slight asymmetry

- The syntax of clients becomes: $a\{(x)P\} \Leftarrow_k Q$
  (we added the name k of the termination handler service to be associated to the session instantiated on the service-side)
- Services are now specified with the process $a \Rightarrow (x)P : (y)T$
  (an additional protocol $(y)T$ is specified which represents the body of a fresh termination handler service that will be associated to the corresponding session on the client-side).

From PSC to SCC - II

Extending sessions

- A service name k, identifying the so-called *termination handler* service, can be associated to each session: $r \rhd_k P$
- The first time the protocol P running inside the session invokes such a service k, the session is closed

Extending services: A slight asymmetry

- The syntax of clients becomes: $a\{(x)P\} \Leftarrow_k Q$
  (we added the name k of the termination handler service to be associated to the session instantiated on the service-side)
- Services are now specified with the process $a \Rightarrow (x)P : (y)T$
  (an additional protocol $(y)T$ is specified which represents the body of a fresh termination handler service that will be associated to the corresponding session on the client-side).

SCC Syntax

Grammar

$P, Q, T, \ldots$	$::= \quad \mathbf{0}$	Nil
	$\mid a.P$	Concretion
	$\mid (x)P$	Abstraction
	$\mid \text{return } a.P$	Return Value
	$\mid a \Rightarrow (x)P : (y)T$	Service Definition
	$\mid a\{(x)P\} \Leftarrow_k Q$	Service Invocation
	$\mid a \rhd_k P$	Session
	$\mid P \mid Q$	Parallel Composition
	$\mid (\nu a)P$	New Name

A special name close is reserved for the specification of session protocols.

Shorthand notation

We write $a \Rightarrow (x)P$ for $a \Rightarrow (x)P : (y)\mathbf{0}$.
We also omit k in $a\{(x)P\} \Leftarrow_k Q$ and $a \Leftarrow_k Q$ when it is not relevant.

SCC Operational Semantics - I

Structural congruence and active contexts

As before (but over the extended syntax).

Termination names

An auxiliary function *tn* is defined on active contexts that keeps track of the *termination names* associated to sessions that enclose the hole:

$$tn([\![\cdot]\!]) = \emptyset \qquad tn(\mathbb{C}|P) = tn(a\{(x)P\} \Leftarrow_s \mathbb{C}) = tn(\mathbb{C})$$
$$tn(a \rhd_s \mathbb{C}) = tn(\mathbb{C}) \cup \{s\} \qquad tn((\nu a)\mathbb{C}) = tn(\mathbb{C}) \setminus \{a\}$$

This function is used to check whether a service invocation should be interpreted as a closing signal for some of the enclosing sessions.

SCC Operational Semantics - II

$$\mathbb{C}[\![\, s \Rightarrow (x)P : (z)T \,]\!] \mid \mathbb{D}[\![\, s\{(y)P'\} \Leftarrow_k (Q|u.R) \,]\!] \;\rightarrow\; (\nu r, k') \left(\begin{array}{l} \mathbb{C}\left[\!\!\left[\begin{array}{l} s \Rightarrow (x)P : (z)T \mid \\ r \rhd_k \left(\begin{array}{l} k' \Rightarrow (z)T\{^k/_{\mathsf{close}}\} \mid \\ P\{^u/_x\}\{^k/_{\mathsf{close}}\} \end{array} \right) \end{array} \right]\!\!\right] \mid \\ \\ \mathbb{D}\left[\!\!\left[\begin{array}{l} \overline{r} \rhd_{k'} P'\{^u/_y\}\{^{k'}/_{\mathsf{close}}\} \mid \\ s\{(y)P'\} \Leftarrow_k (Q|R) \end{array} \right]\!\!\right] \end{array} \right)$$

if $s \notin tn(\mathbb{D})$, r, k' are fresh and u, s, k not bound by $\mathbb{C}$,$\mathbb{D}$

$$r \rhd_s \mathbb{D}[\![\, s\{(y)P\} \Leftarrow_k (Q|u.R) \,]\!] \;\rightarrow\; s\{\} \Leftarrow_k u$$

if $s \notin tn(\mathbb{D})$ and u, s, k not bound by $\mathbb{D}$

$$\mathbb{C}[\![\, r \rhd_k (P|u.Q) \,]\!] \mid \mathbb{D}[\![\, \overline{r} \rhd_{k'} (R|(z)S) \,]\!] \;\rightarrow\; \mathbb{C}[\![\, r \rhd_k (P|Q) \,]\!] \mid \mathbb{D}[\![\, \overline{r} \rhd_{k'} (S\{^u/_z\} \mid R) \,]\!]$$

if u, r not bound by $\mathbb{C}$,$\mathbb{D}$

$$r \rhd_k (P|\mathsf{return}\, u.Q) \;\rightarrow\; u \mid r \rhd_k (P|Q)$$

$$\mathbb{C}[\![\, P \,]\!] \;\rightarrow\; \mathbb{C}[\![\, P' \,]\!] \text{ if } P \equiv Q,\; Q \rightarrow Q',\; Q' \equiv P'$$

SCC Examples: Closure Protocol

A typical usage of termination handler services is the closure of the current session .

A typical service-side closure protocol

$$s \Rightarrow (x)P' : (y)\mathsf{close}\,\{\} \Leftarrow y$$

A typical client-side closure protocol

$$\mathit{End} \triangleq \mathsf{close}\,\{\} \Leftarrow (\mathit{end} \Rightarrow (x)\mathsf{return}\,x))$$

End is designed to be included in the client-side protocol:

$$(\nu\, \mathit{end})s\{\,(y)\big(P \mid \mathit{End}\big)\,\} \Leftarrow_{\mathit{end}} v$$

Closing the client-side session will in turn activate the service-side termination handler.

SCC Examples: Closure Protocol

A typical usage of termination handler services is the closure of the current session .

A typical service-side closure protocol

$$s \Rightarrow (x)P' : (y)\mathsf{close}\,\{\} \Leftarrow y$$

A typical client-side closure protocol

$$End \triangleq \mathsf{close}\,\{\} \Leftarrow (end \Rightarrow (x)\mathsf{return}\,x))$$

End is designed to be included in the client-side protocol:

$$(\nu\, end)s\{\,(y)(P \mid End)\,\} \Leftarrow_{end} v$$

Closing the client-side session will in turn activate the service-side termination handler.

SCC Examples: Closure Protocol

A typical usage of termination handler services is the closure of the current session .

A typical service-side closure protocol

$$s \Rightarrow (x)P' : (y)\mathsf{close}\,\{\} \Leftarrow y$$

A typical client-side closure protocol

$$\mathit{End} \triangleq \mathsf{close}\,\{\} \Leftarrow (\mathit{end} \Rightarrow (x)\mathsf{return}\,x))$$

End is designed to be included in the client-side protocol:

$$(\nu\, \mathit{end})s\{\,(y)\big(P \mid \mathit{End}\big)\,\} \Leftarrow_{\mathit{end}} v$$

Closing the client-side session will in turn activate the service-side termination handler.

SCC Examples: Service Update

Soccer world champion

$$SWC \Rightarrow (-)\texttt{brasil}$$

When a team becomes the new world champion then the service must be updated!

In PSC there is no way to cancel a definition and replace it with a new one.

By contrast, in SCC we can define the termination handler

$$new \Rightarrow (z)\big(\ SWC \Rightarrow (-)z \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return}\, y)\ \big)$$

to be run in parallel with

$$r \rhd_{new} \big(\ SWC \Rightarrow (-)\texttt{brasil} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return}\, y)\ \big)$$

For example, consider the recent invocation

$$update\{\} \Leftarrow \texttt{italy}$$

SCC Examples: Service Update

Soccer world champion

$$SWC \Rightarrow (-)\texttt{brasil}$$

When a team becomes the new world champion then the service must be updated!

In PSC there is no way to cancel a definition and replace it with a new one.

By contrast, in SCC we can define the termination handler

$$new \Rightarrow (z)\big(\ SWC \Rightarrow (-)z \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return}\, y)\ \big)$$

to be run in parallel with

$$r \rhd_{new} \big(\ SWC \Rightarrow (-)\texttt{brasil} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return}\, y)\ \big)$$

For example, consider the recent invocation

$$update\{\} \Leftarrow \texttt{italy}$$

SCC Examples: Service Update - II

And the winner is...

$$\left(\begin{array}{ll} & update\{\} \Leftarrow \texttt{italy} \\ | & new \Rightarrow (z)(\ SWC \Rightarrow (-)z \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & r \rhd_{new} (\ SWC \Rightarrow (-)\texttt{brasil} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid \bar{a} \rhd \mathbf{0} \mid new \Rightarrow (z)(\ \ldots) \\ | & r \rhd_{new} (\ \ldots \mid new\{\} \Leftarrow_{new} (\ \ldots \mid a \rhd \text{return } \texttt{italy}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ \ldots) \\ | & r \rhd_{new} (\ \ldots \mid new\{\} \Leftarrow_{new} (\ \ldots \mid \texttt{italy} \mid a \rhd \mathbf{0}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\ update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ \ldots) \mid new\{\} \Leftarrow_{new} \texttt{italy}\ ) \quad \rightarrow$$

$$(\nu b)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ \ldots) \\ | & b \rhd_{new} (\ SWC \Rightarrow (-)\texttt{italy} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & new\{\} \Leftarrow_{new} \mathbf{0} \mid \bar{b} \rhd \mathbf{0} \end{array}\right)$$

SCC Examples: Service Update - II

And the winner is...

$$\left(\begin{array}{ll} & update\{\} \Leftarrow \texttt{italy} \\ | & new \Rightarrow (z)(\ SWC \Rightarrow (-)z \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & r \rhd_{new} (\ SWC \Rightarrow (-)\texttt{brasil} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid \bar{a} \rhd \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & r \rhd_{new} (\ ... \mid new\{\} \Leftarrow_{new} (\ ... \mid a \rhd \text{return } \texttt{italy}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & r \rhd_{new} (\ ... \mid new\{\} \Leftarrow_{new} (\ ... \mid \texttt{italy} \mid a \rhd \mathbf{0}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\ update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \mid new\{\} \Leftarrow_{new} \texttt{italy}\ ) \quad \rightarrow$$

$$(\nu b)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & b \rhd_{new} (\ SWC \Rightarrow (-)\texttt{italy} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & new\{\} \Leftarrow_{new} \mathbf{0} \mid \bar{b} \rhd \mathbf{0} \end{array}\right)$$

SCC Examples: Service Update - II

And the winner is...

$$\left(\begin{array}{ll} & update\{\} \Leftarrow \texttt{italy} \\ | & new \Rightarrow (z)(\ SWC \Rightarrow (-)z \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & r \rhd_{new} (\ SWC \Rightarrow (-)\texttt{brasil} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid \bar{a} \rhd \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & r \rhd_{new} (\ ... \mid new\{\} \Leftarrow_{new} (\ ... \mid a \rhd \text{return } \texttt{italy}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & r \rhd_{new} (\ ... \mid new\{\} \Leftarrow_{new} (\ ... \mid \texttt{italy} \mid a \rhd \mathbf{0}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\ update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \mid new\{\} \Leftarrow_{new} \texttt{italy}\ ) \quad \rightarrow$$

$$(\nu b)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & b \rhd_{new} (\ SWC \Rightarrow (-)\texttt{italy} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & new\{\} \Leftarrow_{new} \mathbf{0} \mid \bar{b} \rhd \mathbf{0} \end{array}\right)$$

SCC Examples: Service Update - II

And the winner is...

$$\left(\begin{array}{ll} & update\{\} \Leftarrow \texttt{italy} \\ | & new \Rightarrow (z)(\ SWC \Rightarrow (-)z \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & r \rhd_{new} (\ SWC \Rightarrow (-)\texttt{brasil} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid \bar{a} \rhd \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & r \rhd_{new} (\ ... \mid new\{\} \Leftarrow_{new} (\ ... \mid a \rhd \text{return } \texttt{italy}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & r \rhd_{new} (\ ... \mid new\{\} \Leftarrow_{new} (\ ... \mid \texttt{italy} \mid a \rhd \mathbf{0}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\ update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \mid new\{\} \Leftarrow_{new} \texttt{italy}\ ) \quad \rightarrow$$

$$(\nu b)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & b \rhd_{new} (\ SWC \Rightarrow (-)\texttt{italy} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & new\{\} \Leftarrow_{new} \mathbf{0} \mid \bar{b} \rhd \mathbf{0} \end{array}\right)$$

SCC Examples: Service Update - II

And the winner is...

$$\left(\begin{array}{ll} & update\{\} \Leftarrow \texttt{italy} \\ | & new \Rightarrow (z)(\ SWC \Rightarrow (-)z \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & r \rhd_{new} (\ SWC \Rightarrow (-)\texttt{brasil} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid \bar{a} \rhd \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & r \rhd_{new} (\ ... \mid new\{\} \Leftarrow_{new} (\ ... \mid a \rhd \text{return } \texttt{italy}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\nu a)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & r \rhd_{new} (\ ... \mid new\{\} \Leftarrow_{new} (\ ... \mid \texttt{italy} \mid a \rhd \mathbf{0}\ )\ ) \end{array}\right)$$

$$\rightarrow \quad (\ update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \mid new\{\} \Leftarrow_{new} \texttt{italy}\ ) \quad \rightarrow$$

$$(\nu b)\left(\begin{array}{ll} & update\{\} \Leftarrow \mathbf{0} \mid new \Rightarrow (z)(\ ...) \\ | & b \rhd_{new} (\ SWC \Rightarrow (-)\texttt{italy} \mid new\{\} \Leftarrow_{new} (update \Rightarrow (y)\text{return } y)\ ) \\ | & new\{\} \Leftarrow_{new} \mathbf{0} \mid \bar{b} \rhd \mathbf{0} \end{array}\right)$$

SCC Examples: Encoding Orc in SCC - I

SCC as a service orchestration language

To evaluate the expressiveness and usability of SCC as a language for service orchestration, one has to challenge its ability of encoding some frequently used service composition patterns.

Workflow patterns

A library of basic patterns, called the *workflow patterns*, has been identified by van der Aalst et al.

Orc can conveniently model most workflow patterns! [Coordination'06]

The Orc challenge

If we can show that SCC can encode Orc, then by transitivity we can implement van der Aalst's workflow patterns.

SCC Examples: Encoding Orc in SCC - I

SCC as a service orchestration language

To evaluate the expressiveness and usability of SCC as a language for service orchestration, one has to challenge its ability of encoding some frequently used service composition patterns.

Workflow patterns

A library of basic patterns, called the *workflow patterns*, has been identified by van der Aalst et al.

Orc can conveniently model most workflow patterns! [Coordination'06]

The Orc challenge

If we can show that SCC can encode Orc, then by transitivity we can implement van der Aalst's workflow patterns.

SCC Examples: Encoding Orc in SCC - I

SCC as a service orchestration language

To evaluate the expressiveness and usability of SCC as a language for service orchestration, one has to challenge its ability of encoding some frequently used service composition patterns.

Workflow patterns

A library of basic patterns, called the *workflow patterns*, has been identified by van der Aalst et al.

Orc can conveniently model most workflow patterns! [Coordination'06]

The Orc challenge

If we can show that SCC can encode Orc, then by transitivity we can implement van der Aalst's workflow patterns.

SCC Examples: Encoding Orc in SCC - II

While a value is trivially encoded as itself, i.e., $[\![u]\!] = u$, for variables (and thus for actual parameters) we need two different encodings, depending on whether they are passed by name or evaluated.

We distinguish the two encodings by different subscripts:

$$[\![x]\!]_n = x \qquad [\![x]\!]_v = x \Leftarrow \bullet$$

- The evaluation of a variable x is encoded as a request for the current value to the variable manager of x.
- Variable managers are created by both sequential composition and asymmetric parallel composition.

SCC Examples: Encoding Orc in SCC - II

$$
\begin{aligned}
[\![E(x) \triangleq P]\!] &= E \Rightarrow (x)[\![P]\!] \\
[\![a(p)]\!] &= a \Leftarrow [\![p]\!]_v \\
[\![x(p)]\!] &= (\nu\, forw, pub)\big(\ forw\{\} \Leftarrow [\![x]\!]_v \mid \\
&\qquad\qquad forw \Rightarrow (a)pub\{\} \Leftarrow [\![a(p)]\!] \mid \\
&\qquad\qquad pub \Rightarrow (y)\mathsf{return}\, y\ \big) \\
[\![E(p)]\!] &= E \Leftarrow [\![p]\!]_n \\
[\![P|Q]\!] &= [\![P]\!]|[\![Q]\!] \\
[\![P > x > Q]\!] &= (\nu\, z, pub)\big(\ z\{\} \Leftarrow [\![P]\!] \mid \\
&\qquad\qquad z \Rightarrow (y)(\nu x)(x \Rightarrow (-)y \mid pub\{\} \Leftarrow [\![Q]\!]) \mid \\
&\qquad\qquad pub \Rightarrow (y)\mathsf{return}\, y\ \big) \\
[\![Q\ \textbf{where}\, x :\in P]\!] &= (\nu\, x, z, w)\big(\ [\![Q]\!] \mid \\
&\qquad\qquad z \Rightarrow (y)(x \Rightarrow (-)y) \mid \\
&\qquad\qquad w\{\} \Leftarrow_z \bullet \mid \\
&\qquad\qquad w \Rightarrow (-)(\mathsf{close}\,\{\} \Leftarrow [\![P]\!])\ \big)
\end{aligned}
$$

From Orc to SCC: An Example

Emailing news in Orc

Let us consider the Orc expression

$$CNN(d)|BBC(d) > x > email(x)$$

which invokes the news services of both *CNN* and *BBC* asking for news of day d. For each reply it sends an email (to a default address) with the received news. Thus this expression can send from zero up to two emails. The SCC encoding is as follows:

$$\begin{aligned}(\nu z, pub)\big(z\{\} \Leftarrow (CNN \Leftarrow d|BBC \Leftarrow d) \mid \\ z \Rightarrow (y)(\nu x)(x \Rightarrow (-)y \mid pub\{\} \Leftarrow email \Leftarrow x \Leftarrow \bullet) \mid \\ pub \Rightarrow (y)\text{return}\, y\ \big)\end{aligned}$$

We have supposed here to have *CNN*, *BBC* and *email* available as services.

Outline

Concluding Remarks

What's new?

The main novelty regards session handling mechanisms for the definition of

- *session naming and scoping*
- *structured interaction protocols*
- *service interruption, cancelation and update* (dynamic environment)

In particular

- The protocols to be run within the service-side / client-side session are well-exposed in the syntax of the calculus to favour type checking, service conformance check, service discovery
- While Orc's cancelation is too demanding (it can destroy a wide area computation), SCC has just a local termination that activates a proper handler at the partner site.

And why not just π?

- Higher-level primitives can favour and make more scalable the development of typing systems and proof techniques.

Future Work

Alternatives and Future extensions

Ongoing discussions about:

- Multi-party sessioning
- Replicator / recursion / return P
- Synchronized termination

Next issues on the stack:

- Distribution
- Types
- Long-running transactions and compensations
- Delegation
- XML querying
- SLA and QoS

THANKS!

$$
\begin{array}{rcl}
Question? & \Rightarrow & (q)selectCoAuthor \Leftarrow q \\
\mid selectCoAuthor & \Rightarrow & (q)Antonio \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)Davide \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)Francisco \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)Gianluigi \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)Ivan \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)Luis \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)MicheleB \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)MicheleL \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)Rocco \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)Ugo \Leftarrow_{selectCoAuthor} q \\
\mid selectCoAuthor & \Rightarrow & (q)Vasco \Leftarrow_{selectCoAuthor} q
\end{array}
$$

PSC Examples: News Streaming

Programming Pattern

The service *pub* (or alike) can be useful in many applications.

In fact, in PSC *sessions cannot be closed* and therefore recursive invocations on the client-side are nested at increasing depth (while the return instruction can move values only one level up).

News Streaming (client side)

A recursive process that repeatedly invokes service s on value x with publishing service p is shown below:

$$rec \Rightarrow (s, x, p)s\left\{(-)\left(\begin{array}{l}(y)p\{\} \Leftarrow y \\ \mid \quad rec\{\} \Leftarrow \langle s, x, p\rangle\end{array}\right)\right\} \Leftarrow x$$

Sample of invocation of the service *rec*:

$$rec\{\} \Leftarrow \langle ANSA, \bullet, pub\rangle \mid pub \Rightarrow (x)\text{return}\, x$$

that returns the stream of news obtained from the *ANSA* service.

PSC Examples: Room Booking

Room Booking (service side)

$$bookRoom \Rightarrow (d)\left(\begin{array}{l} avail \Leftarrow d \mid \\ (cs)(\nu\, code)code.(cc)epay\{(-)cc.(i)\text{return } i\} \Leftarrow price \Leftarrow cs \end{array}\right)$$

Room Booking (client side)

$$bookRoom\{(-)(r)(select \Leftarrow r \mid (c)myCCnum.(cid)\text{return } \langle c, cid\rangle)\} \Leftarrow \texttt{dates}$$

Comments

bookRoom is invoked with the dates *d* for the reservation, it gets (from the local service *avail*) and passes to the client the set of available rooms. The client sends her selection *cs*. A fresh reservation code is sent to the client. The client sends her credit card number *cc*. The service debits the cost to the credit card (via a suitable electronic payment service *epay*). Finally, if everything is ok, the client receives the confirmation id *i* generated by *epay*.
Note that we suppose a service *select* for interacting with the user and *price* that computes the price of the chosen room.

PSC Examples: Encoding of the lazy λ-calculus

The translation is in the spirit of Milner's π-calculus encoding:

$$
\begin{array}{rcl}
[\![x]\!]_p & = & x\{\} \Leftarrow p \\
[\![\lambda x.M]\!]_p & = & p \Rightarrow (x)(q)[\![M]\!]_q \\
[\![M\,N]\!]_p & = & (\nu m)(\nu n)\left(\begin{array}{ll} & [\![M]\!]_m \\ | & n \Rightarrow (s)[\![N]\!]_s \\ | & m\{(-)p\} \Leftarrow n \end{array}\right)
\end{array}
$$

The more important differences

1. each service invocation opens a new session where the computation can progress (remind that sessions cannot be closed in PSC)
2. all service definitions will remain available even when no further invocation will be possible.

If on one hand, the encoding witnesses the expressive power of PSC, on the other hand, it also motivates the introduction of some mechanism for closing sessions.

Encoding of PSC into π-calculus

The encoding below shows that PSC can be seen as a fragment of the π-calculus.

$$
\begin{aligned}
[\![a\{(x)P\} \Leftarrow Q]\!]_{in,out,ret} &= (\nu z)\big(\ [\![Q]\!]_{in,z,ret} \mid !z(x).(\nu r,\tilde{r})\overline{a}\langle r,\tilde{r},x\rangle.[\![P]\!]_{r,\tilde{r},out}\ \big)\\
[\![a \Rightarrow (x)P]\!]_{in,out,ret} &= !a(r,\tilde{r},x).([\![P]\!]_{\tilde{r},r,out})\\
[\![a \rhd P]\!]_{in,out,ret} &= [\![P]\!]_{a,\tilde{a},out}\\
[\![a.P]\!]_{in,out,ret} &= \overline{out}\ a.[\![P]\!]_{in,out,ret}\\
[\![(x)P]\!]_{in,out,ret} &= in(x).[\![P]\!]_{in,out,ret}\\
[\![\text{return } a.P]\!]_{in,out,ret} &= \overline{ret}\ a \mid [\![P]\!]_{in,out,ret}\\
[\![P \mid Q]\!]_{in,out,ret} &= [\![P]\!]_{in,out,ret} \mid [\![Q]\!]_{in,out,ret}\\
[\![(\nu x)P]\!]_{in,out,ret} &= (\nu x)[\![P]\!]_{in,out,ret}\\
[\![\mathbf{0}]\!]_{in,out,ret} &= \mathbf{0}
\end{aligned}
$$

The encoding can hardly be extended to full SCC calculus due to the session interruption mechanism that has no direct couterpart in the π-calculus.

SCC Examples: A blog service - I

Blog

We consider a service that implements a *blog*, i.e. a web page used by a web client to log personal annotations.

Interaction with the Blog

A blog provides two services:

- *get* to read the current contents of the blog
- *set* to modify the contents.

The close-free fragment is not expressive enough to faithfully model such a service because it does not support service update, here needed to update the blog contents.

SCC Examples: A blog service - II

Blog Factory

$$
\begin{array}{l}
\mathit{newBlog} \Rightarrow (v, \mathit{get}, \mathit{set})\big(\mathit{blog}\{\} \Leftarrow_{\mathit{newBlog}} \langle v, \mathit{get}, \mathit{set}\rangle\big) \mid \\
\mathit{blog} \Rightarrow (v, \mathit{get}, \mathit{set})\big(\quad \mathit{get} \Rightarrow (-)v \mid \\
\qquad\qquad\qquad\qquad\quad \mathsf{close}\,\{\} \Leftarrow (\mathit{set} \Rightarrow (v')\mathsf{return}\,\langle v', \mathit{get}, \mathit{set}\rangle) \quad \big)
\end{array}
$$

We use the service *newBlog* as a factory of blogs. It receives three names:

- the initial contents v
- the name for the new *get* service
- the name for the new *set* service

Upon invocation, the factory forwards the three received values to the *blog* service which is the responsible for the actual instantiation of the *get* and *set* services.

The update of the blog contents is achieved by invoking the service close which is bound to *newBlog*; this invocation cancels the currently available *get* and *set* services and delegates to *newBlog* the creation of their new instances passing also the updated contents v'.

SCC Examples: A blog service - III

Blog update

The process below installs a wiki page with initial contents v, then it adds some new contents v'.

$$newBlog\{\} \Leftarrow \langle v, get, set\rangle \mid$$
$$set\{\} \Leftarrow \big(concat\{\,(-)v'.get \Leftarrow \bullet \mid (x)\text{return}\,x\,\} \Leftarrow \bullet\big)$$

The service *concat* simply computes the new contents appending v' to the contents v received after service invocation:

$$concat \Rightarrow (-)(y)(z).y \circ z$$

Here $\circ$ denotes justaposition of blog contents.